Gene order rearrangement methods for the reconstruction of phylogeny

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet
Informatik

vorgelegt

von Dipl.-Inf. Matthias Bernt
geboren am 16. Januar 1979 in Gera

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Martin Middendorf, Universität Leipzig
2. Dr. Marie-France Sagot, INRIA und Universität Lyon

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung
am 29. Januar 2010 mit dem Gesamtprädikat summa cum laude.

Acknowledgements

First of all I would like to thank my advisor Prof. Dr. Martin Middendorf for providing me the excellent and inspiring scientific environment and the ongoing support, guidance, advice, and time that enabled me to write this thesis. I also have to thank Prof. Dr. Daniel Merkle who has been a source of inspiration for me.

I have to thank Prof. Dr. Peter F. Stadler and the Bioinformatics group at the University of Leipzig for all the discussions we had throughout the fruitful ongoing cooperation, and especially Dr. Guido Fritzsch for giving me guidance behind the scenes of Biology.

I thank Dr. Marie-France Sagot for kindly reviewing this thesis.

I am much obliged for the possibility to visit research groups around the world. Most of all I have to thank Prof. Kun-Mao Chao and his research group at the National Taiwan University in Taipei for providing a very nice and productive working environment. Furthermore, I am very grateful for the cordial welcome, the introduction to Taiwanese culture, and for showing me the most beautiful places Taiwan. 谢谢. Additionally I have to thank Dr. Eric Tannier and Dr. Marie-France Sagot at the University of Lyon as well as Dr. Jean-Stéphane Varré and Dr. Hélène Touzet at the University of Lille for hosting me and the cooperation which has been very informative and inspiring for me.

For the discussions, help, and pleasant working environment I have to thank my colleagues at the Parallel Computing and Complex Systems group: Dr. Stefan Janson, Dr. Sebastian Lange, Kai Ramsch, Konrad Diwold, Stephanie Keller-Schmidt, Alexander Scheidler, and Nicolas Wieseke.

A gigantic thank you to Annekatrin Roth, Nicolas Wieseke, Prof. Dr. Daniel Merkle, Dr. Guido Fritzsch, and Dr. Sebastian Lange for proofreading the manuscript. An additional enormous thank you to Annekatrin Roth for all the support, encouragement, and distraction throughout the time.

Above all I want to thank my family for all the support on my way.

My work has been supported financially by the German Research Foundation (DFG) through the DFG Graduiertenkolleg Wissensrepräsentation (Research Training Unit "Knowledge representation") at the University of Leipzig and the priority programme 1174 "Deep

Metazoan Phylogeny“. The exchange program with the NTU in Taipei was funded by the German Academic Exchange Service (DAAD) within grant D/05/06868.

Contents

1 Introduction

The study of phylogeny, i.e. the evolutionary history of species, is a central problem in biology and a key for understanding characteristics of contemporary species. Many problems in this area can be formulated as combinatorial optimisation problems which makes it particularly interesting for computer scientists. The reconstruction of the phylogeny of species can be based on various kinds of data, e.g. morphological properties or characteristics of the genetic information of the species. Maximum parsimony is a popular and widely used method for phylogenetic reconstruction aiming for an explanation of the observed data requiring the least evolutionary changes.

A certain property of the genetic information gained much interest for the reconstruction of phylogeny in recent time: the organisation of the genomes of species, i.e. the arrangement of the genes on the chromosomes. But the idea to reconstruct phylogenetic information from gene arrangements has a long history. In Dobzhansky and Sturtevant (1938) it was already pointed out that "a comparison of the different gene arrangements in the same chromosome may, in certain cases, throw light on the historical relationships of these structures, and consequently on the history of the species as a whole". This kind of data is promising for the study of deep evolutionary relationships because gene arrangements are believed to evolve slowly (Rokas and Holland, 2000). This seems to be the case especially for mitochondrial genomes which are available for a wide range of species (Boore, 1999).

The development of methods for the reconstruction of phylogeny from gene arrangement data has made considerable progress during the last years. Prominent examples are the computation of parsimonious evolutionary scenarios, i.e. a shortest sequence of rearrangements transforming one arrangement of genes into another or the length of such a minimal scenario (Hannenhalli and Pevzner, 1995b; Sankoff, 1992; Watterson et al., 1982); the reconstruction of parsimonious phylogenetic trees from gene arrangement data (Bader et al., 2008; Bernt et al., 2007b; Bourque and Pevzner, 2002; Moret et al., 2002a); or the computation of the similarities of gene arrangements (Bergeron et al., 2008a; Heber et al., 2009).

The central theme of this work is to provide efficient algorithms for modified versions of fundamental genome rearrangement problems using more plausible rearrangement models. Two types of modified rearrangement models are explored.

The first type is to restrict the set of allowed rearrangements as follows. It can be observed that certain groups of genes are preserved during evolution. This may be caused by functional constraints which prevented the destruction (Lathe et al., 2000; Sémon and Duret, 2006; Xie et al., 2003), certain properties of the rearrangements which shaped the gene orders (Eisen et al., 2000; Sankoff, 2002; Tillier and Collins, 2000), or just because no destructive rearrangement happened since the speciation of the gene orders. It can be assumed that gene groups, found in all studied gene orders, are not acquired independently. Accordingly, these gene groups should be preserved in plausible reconstructions of the course of evolution, in particular the gene groups should be present in the reconstructed putative ancestral gene orders. This can be achieved by restricting the set of rearrangements, which are allowed for the reconstruction, to those which preserve the gene groups of the given gene orders. Since it is difficult to determine functionally what a gene group is, it has been proposed to consider common combinatorial structures of the gene orders as gene groups (Marcotte et al., 1999; Overbeek et al., 1999).

The second considered modification of the rearrangement model is extending the set of allowed rearrangement types. Different types of rearrangement operations have shuffled the gene orders during evolution. It should be attempted to use the same set of rearrangement operations for the reconstruction otherwise distorted or even wrong phylogenetic conclusions may be obtained in the worst case.

Both possibilities have been considered for certain rearrangement problems before. Restricted sets of allowed rearrangements have been used successfully for the computation of parsimonious rearrangement scenarios consisting of inversions only where the gene groups are identified as common intervals (Bérard et al., 2007; Figeac and Varré, 2004). Extending the set of allowed rearrangement operations is a delicate task. On the one hand it is unknown which rearrangements have to be regarded because this is part of the phylogeny to be discovered. On the other hand, efficient exact rearrangement methods including several operations are still rare, in particular when transpositions should be included. For example, the problem to compute shortest rearrangement scenarios including transpositions is still of unknown computational complexity. Currently, only efficient approximation algorithms are known (e.g. Bader and Ohlebusch, 2007; Elias and Hartman, 2006).

Two problems have been studied with respect to one or even both of these possibilities in the scope of this work.

The first one is the inversion median problem. Given the gene orders of some taxa, this problem asks for potential ancestral gene orders such that the corresponding inversion scenario is parsimonious, i.e. has a minimum length. Solving this problem is an essential component

of algorithms for computing phylogenetic trees from gene arrangements (Bourque and Pevzner, 2002; Moret et al., 2002a, 2001). The unconstrained inversion median problem is NP-hard (Caprara, 2003). In Chapter 3 the inversion median problem is studied under the additional constraint to preserve gene groups of the input gene orders. Common intervals, i.e. sets of genes that appear consecutively in the gene orders, are used for modelling gene groups. The problem of finding such ancestral gene orders is called the preserving inversion median problem. Already the problem of finding a shortest inversion scenario for two gene orders is NP-hard (Figeac and Varré, 2004).

Mitochondrial gene orders are a rich source for phylogenetic investigations because they are known for more than 1 000 species. Four rearrangement operations are reported at least in the literature to be relevant for the study of mitochondrial gene order evolution (Boore, 1999): That is inversions, transpositions, inverse transpositions, and tandem duplication random loss (TDRL). Efficient methods for a plausible reconstruction of genome rearrangements for mitochondrial gene orders using all four operations are presented in Chapter 4.

An important rearrangement operation, in particular for the study of mitochondrial gene orders, is the tandem duplication random loss operation (e.g. Boore, 2000; Mauro et al., 2006). This rearrangement duplicates a part of a gene order followed by the random loss of one of the redundant copies of each gene. The gene order is rearranged depending on which copy is lost. This rearrangement should be regarded for reconstructing phylogeny from gene order data. But the properties of this rearrangement operation have rarely been studied (Bouvel and Rossin, 2009; Chaudhuri et al., 2006). The combinatorial properties of the TDRL operation are studied in Chapter 5. The enumeration and counting of sorting TDRLs, that is TDRL operations reducing the distance, is studied in particular. Closed formulas for computing the number of sorting TDRLs and methods for the enumeration are presented. Furthermore, TDRLs are one of the operations considered in Chapter 4. An interesting property of this rearrangement, distinguishing it from other rearrangements, is its asymmetry. That is the effects of a single TDRL can (in the most cases) not be reversed with a single TDRL. The use of this property for phylogeny reconstruction is studied in Section 4.3.

This thesis is structured as follows. The existing approaches obeying similar types of modified rearrangement models as well as important concepts and computational methods to related problems are reviewed in Chapter 2. The combinatorial structures of gene orders that have been proposed for identifying gene groups, in particular common intervals, as well as the computational approaches for their computation are reviewed in Section 2.2. Approaches for computing parsimonious pairwise rearrangement scenarios are outlined in Section 2.3. Methods for the computation genome rearrangement scenarios obeying biologically motivated constraints, as introduced above, are detailed in Section 2.4. The approaches for the inversion median problem are covered in Section 2.5. Methods for the reconstruction of phylogenetic trees from gene arrangement data are briefly outlined in Section 2.6.

Chapter 3 introduces the new algorithms CIP, ECIP, and TCIP for solving the preserving inversion median problem. The efficiency of the algorithm is empirically studied for simulated as well as mitochondrial data. The description of algorithms CIP and ECIP is based on Bernt et al. (2006b). TCIP has been described in Bernt et al. (2007a, 2008b). But the theoretical foundation of TCIP is extended significantly within this work in order to allow for more than three input permutations.

Gene order rearrangement methods that have been developed for the reconstruction of the phylogeny of mitochondrial gene orders are presented in the fourth chapter. The presented algorithm `CREx` computes rearrangement scenarios for pairs of gene orders. `CREx` regards the four types of rearrangement operations which are important for mitochondrial gene orders. Based on `CREx` the algorithm `TreeREx` for assigning rearrangement events to a given tree is developed. The quality of the `CREx` reconstructions is analysed in a large empirical study for simulated gene orders. The results of `TreeREx` are analysed for several mitochondrial data sets. Algorithms `CREx` and `TreeREx` have been published in Bernt et al. (2008a, 2007c). The analysis of the mitochondrial gene orders of *Echinodermata* was included in Perseke et al. (2008). Additionally, a new and simple method is presented to explore the potential of the `CREx` method. The new method is applied to the complete mitochondrial data set.

The problem of enumerating and counting sorting TDRLs is studied in Chapter 5. The theoretical results are covered to a large extent by Bernt et al. (2009b). The missing combinatorial explanation for some of the presented formulas is given here for the first time. Therefor, a new method for the enumeration and counting of sorting TDRLs has been developed (Bernt et al., 2009a).

2 Related work

2.1 Gene arrangements and genome rearrangements

2.1.1 The biological context

The blueprint for the development and operation of living beings is stored as sequence composed of four types of nucleotides in double stranded molecules called DNA. The whole entirety of DNA is referred to as genome which is organised in one or more molecules called chromosomes. The number of these organisational units differs between species. One speaks of multichromosomal or unichromosomal genomes. Furthermore, the chromosomes are either linear or circular. Stretches of the DNA coding for certain functions are called genes. Such a stretch encodes the information to build one of the myriad of the organism's components, i.e. on the one hand different proteins or RNAs, and on the other hand such a stretch of DNA may regulate the intricate mechanisms building the organism. The genetic information of species is subject to various kinds of mutations affecting the organism's potential to function in the world and create offspring. In the long term some mutations are selected to be passed on through the generations and others are not. Mutation and selection are the ongoing driving forces which generated the rich diversity of species in which mankind lives nowadays.

Besides the classification of mutations by the effect on the fitness of species in nature, mutations can also be categorised by the way the genetic information is modified. There are local mutations replacing, deleting, or inserting single nucleotides and there are non-local mutations rearranging the genome. These mutations modify the arrangement of genes on chromosomes, e.g. moving genes to a different position in the genome or to a different strand. The genomes are not reordered arbitrarily during evolution, but certain types of rearrangement operations are assumed. For example the following four rearrangement operations can be assumed for mitochondrial genomes:

1. Inversions reverse a continuous part of the chromosome, i.e. the order of the affected genes is reversed and they are moved to the other strand.

2. Transpositions move a continuous part of the chromosome to a distant position.

3. Inverse transpositions transpose a continuous part of the chromosome to a distant position where is inserted inversely.

4. A tandem duplication random loss is a tandem duplication of a continuous part of a chromosome followed by the random loss of one copy of each redundant gene.

Other types of rearrangement are not covered by this work, e.g. modification of the chromosome number by fission or fusion of chromosomes, modifications of the gene content by insertion or deletion of genes, and the exchange of genetic material between species.

The reconstruction of phylogeny is to draw conclusions from properties of contemporary species to their evolutionary relationship. Different properties can be regarded for this task, e.g. morphology, the sequence of a certain gene, and the order of the genes which is considered here in this work. The focus of this work lies on rearrangements of circular or linear unichromosomal genomes containing one copy of each gene. It is assumed that the genomes which are compared contain the same set of genes. With this assumptions gene arrangements are correctly modelled by permutations. Each element of the permutations represents one of the considered genetic markers in the chromosomes, usually a gene. An additional sign added to each element represents the strandedness of the corresponding genetic marker. See Figure 2.1 for examples of three metazoan gene orders.

With a few exceptions these assumptions apply for example to mitochondrial genomes (Boore, 1999) but they are too restrictive to allow for the study of the genomes with unequal gene content, duplicated genes, or multichromosomal genomes.

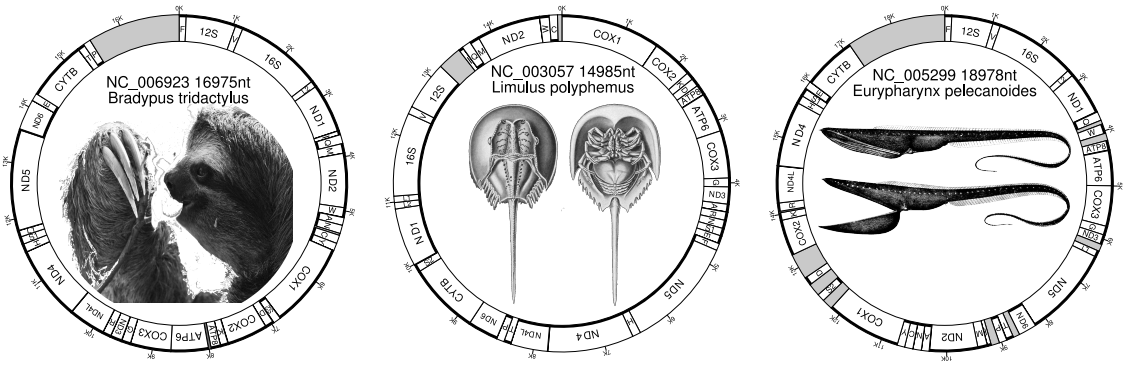


Figure 2.1 – Representation of the circular mitochondrial genomes of three *metazoan* species drawn with mtviz (`http://pacosy.informatik.uni-leipzig.de/mtviz`); gene names are written inside the block representing the gene; the strandedness is represented by a thick line (The image of Bradypus is published under Creative Commons licence; the image of *Limulus* taken from Haeckel (1914) "Kunstformen der Natur" image of *Eurypharynx* taken from Goode and Bean (1896) "Oceanic Ichthyology")

2.1.2 The formal background

In the following the formal background and the notations used throughout this work are introduced.

Assuming that each gene occurs exactly once in each genome of species and genes are organised in a linear fashion on a single chromosome, gene arrangements can be modelled as permutations. A *permutation* $\pi = (\pi(1)\ \pi(2)\ \ldots\ \pi(n))$ of size n is a permutation of the set $\{1, \ldots, n\}$, i.e. a sequence of elements such that each element appears exactly once. The i-th element of a permutation of size n is denoted by $\pi(i)$, with $i \in [1 : n]$. The *inverted permutation* $-\pi = (\pi(n)\ \ldots\ \pi(1))$. This is not to be confused with the *inverse permutation* π^{-1} of π is defined as the permutation of the same elements such that $\pi^{-1}(e) = i$ iff $\pi(i) = e$ for an element e of π at position i. That is the inverse permutation gives the positions of the elements of a permutation. Recall, $\pi \circ \pi^{-1} = \iota$ holds, where $\circ$ denotes the composition operation. A *signed permutation* is a permutation where each element has an additional sign + or − indicating the strandedness of the genes. The sign + is usually omitted. If the context is clear, the term permutation is used instead of signed permutation. In order to avoid confusion, the j-th permutation in a sequence of permutations Π is denoted by π_j. That is $\pi_j(i)$ denotes the i-the element of the j-th permutation. The identity permutation of size n is given by $\iota = (1\ 2\ \ldots\ n)$. The size of the identity may be omitted when the context is clear. An *interval* of a permutation is a set of consecutive (unsigned) elements of this permutation.

As pointed out above, chromosomes are not linear in general but also circular chromosomes have to be considered. The gene order of circular genomes can be represented as circular (signed) permutations. But usually they are also represented as linear (signed) permutations by cutting them at some position. In this case special care has to be taken because the linear order is only one representative. Hence, the linear representative has to be regarded as equivalent to circular shifts and their the inverted gene orders the circular shifts.

Linear representatives of the circular gene orders shown in Figure 2.1 are:

Bradypus

```
COX1 -S2 D COX2 K ATP8 ATP6 COX3 G ND3 R ND4L ND4 H S1 L1 ND5 -ND6 -E CYTB T
-P F 12S V 16S L2 ND1 I -Q M ND2 W -A -N -C -Y
```

Limulus

```
COX1 COX2 K D ATP8 ATP6 COX3 G ND3 A R N S1 E -F -ND5 -H -ND4 -ND4L T -P ND6
CYTB S2 -ND1 -L2 -L1 -16S -V -12S I -Q M ND2 W -C -Y
```

Eurypharynx

```
COX1 -S2 D COX2 K R ND4L ND4 H S1 -E CYTB F 12S V 16S L2 ND1 -Q W ATP8 ATP6
COX3 G ND3 L1 ND5 -ND6 T -P I M ND2 -A -N -C -Y
```

For linear gene orders two possibilities do exist for defining their equivalence. When complete chromosomes are compared, the orientation of the chromosome does not matter and the inverse of a gene order is considered as identical. In this case the gene orders (respectively permutations) are called *unoriented*. Otherwise, if for example only a part of a chromosome is considered, the orientation of this part in the chromosome is important. In this case the gene orders (respectively permutations) are denoted as *oriented*.

The four considered rearrangements are defined formally as follows.

1. An *inversion* $\rho_{\mathrm{I}}(i,j)$, with $1 \leq i \leq j \leq n$, applied to a signed permutation π of size n transforms it into $\pi \circ \rho_{\mathrm{I}}(i,j) = (\pi(1) \ \ldots \ \pi(i-1) \ -\pi(j) \ \ldots \ -\pi(i) \ \pi(j+1) \ \ldots \ \pi(n))$.

2. A *transposition* $\rho_{\mathrm{T}}(i,j,k)$, $1 \leq i \leq j < k \leq n$ applied to π transforms it into $\pi \circ \rho_{\mathrm{T}}(i,j,k) = (\pi(1) \ \ldots \ \pi(i-1) \ \pi(j+1) \ \ldots \ \pi(k) \ \pi(i) \ \ldots \ \pi(j) \ \pi(k+1) \ \ldots \ \pi(n))$.

3. An *inverse transposition* $\rho_{\mathrm{iT}}(i,j,k)$, with $1 \leq i \leq j \leq n$ and $j < k \leq n$, respectively $1 \leq k < i$, applied to π transforms it into
   $\pi \circ \rho_{\mathrm{iT}}(i,j,k) = (\pi(1) \ \ldots \ \pi(i-1) \ \pi(k) \ \ldots \ -\pi(j+1) \ \pi(i) \ \ldots \ \pi(j) \ \pi(k+1) \ \ldots \ \pi(n))$,
   respectively
   $\pi \circ \rho_{\mathrm{iT}}(i,j,k) = (\pi(1) \ \ldots \ \pi(k-1) \ \pi(i) \ \ldots \ \pi(j) \ -\pi(i-1) \ \ldots \ -\pi(k) \ \pi(j+1) \ \ldots \ \pi(n))$.

4. A *tandem duplication* duplicates an interval of a permutation in tandem, i.e. the two copies of the interval are adjacent in the resulting gene order. Note, the result of a tandem duplication is no permutation. A *tandem duplication random loss* ρ_{TDRL} duplicates a continuous segment of genes in tandem, followed by the loss of one copy of each of the duplicated genes. Formally, a TDRL applied to a permutation π is defined as $\rho_{\mathrm{TDRL}}(F,S)$, where F specifies the set of elements which are kept in the first copy and S defines the set of elements kept in the second copy, such that i) $F \subseteq \{1,\ldots,n\}$ and $S \subseteq \{1,\ldots,n\}$, ii) $F \cup S$ is an interval in π, and iii) $F \cap S = \emptyset$.

Note, inversions are called often *reversal* in the computer science literature. Throughout this work the term inversion is used that is common to the biology literature.

Alternative to the specification of the rearrangements by indices, they can be defined by the set or sets of elements that are affected:

1. $\rho_{\mathrm{I}}(i,j) = \rho_{\mathrm{I}}(\{\pi(i),\ldots,\pi(j)\})$,

2. $\rho_{\mathrm{T}}(i,j,k) = \rho_{\mathrm{T}}(\{\{\pi(i),\ldots,\pi(j)\},\{\pi(j+1),\ldots,\pi(k)\}\})$, and

3. $\rho_{\mathrm{iT}}(i,j,k) = \rho_{\mathrm{iT}}((\{\pi(j+1),\ldots,\pi(k)\},\{\pi(i),\ldots,\pi(j)\}))$.

Transpositions are treated as set of size two containing the sets of transposed elements and inverse transpositions as pair of sets where the first element of the pair specifies the elements

which are additionally inverted. A *rearrangement scenario* (for short scenario) for two signed permutations π and σ is a sequence of rearrangement operations transforming π into σ. A sequence with a minimal number of operations is called *parsimonious* or sorting.

2.2 Gene clusters

The computational analysis of clusters of genes that have been preserved during evolution has gained lots of attention during the last decade. Such clusters are usually defined as sets of genes fulfilling some proximity constraint. This is motivated by the observation that certain gene groups are preserved during evolution. There are diverse possible reasons for the persistence of a gene group, e.g. genes which are co-expressed (Sémon and Duret, 2006), operons (subsequent genes transcribed in a single mRNA) (Lathe et al., 2000; Xie et al., 2003), the prevalence of short rearrangements (Sankoff, 2002), or restrictions of the replication mechanism of the genomes (Eisen et al., 2000; Tillier and Collins, 2000). Of course its also possible that a cluster of genes was not separated or emerged by chance.

There are different interesting aspects, e.g. identifying functionally related gene groups as genes which are in close proximity in a set of gene orders (see following sections for approaches), computing phylogenetic trees and ancestral gene arrangements from the preserved gene clusters (see Section 2.6.2), or the computation of rearrangement scenarios, which regard the information on conserved gene clusters (see Section 2.4). The computation of gene cluster preserving rearrangement scenarios is also a main theme of this work. Therefore, the relevant computational approaches for gene cluster detection will be reviewed in the following.

2.2.1 Conserved adjacencies and breakpoints

Conserved adjacencies and their counterpart breakpoints are the most basic approach to capture the similarities and dissimilarities of gene orders. Formally, given two (unsigned) permutations π and σ of size n. An adjacent pair of elements (e, f) of π is called a *breakpoint* in π (relative to σ) iff neither (e, f) nor (f, e) appear consecutively in that order in σ, otherwise it is called *conserved adjacency*. If π and σ are signed permutations, the definition has to be altered slightly. An adjacency (e, f) of π is called *breakpoint* if neither (e, f) nor $(-f, -e)$ appear consecutively in σ and otherwise *conserved adjacency*. The computation of the breakpoints and conserved adjacencies is straightforward and can be accomplished in linear time.

The number of breakpoints of two permutations π and σ, called *breakpoint distance*, (Blanchette et al., 1999; Sankoff and Blanchette, 1998) denoted by $bp(\pi, \sigma)$, can be used as a measure of dissimilarity. Similarly, the number of conserved adjacencies is a measure for the similarity of two permutations. The notion of conserved adjacencies can be

generalised to sets of permutations by demanding that the adjacency is found in all permutations. In this way the similarity of sets of permutations can be evaluated.

Example 2.2.1. *The signed permutation* π = (1 2 -5 -4 3) *has two conserved adjacencies with respect to the identity permutation:* $(1, 2)$ *and* $(-5, -4)$. *This is because* $(1, 2)$ *and* $(4, 5)$ *are also adjacent in the identity. The other two adjacencies of* π*, i.e.* $(2, -5)$ *and* $(-4, 3)$*, are breakpoints with respect to the identity because elements* 2 *and* 5 *are not adjacent in the identity and elements* 3 *and* 4 *are adjacent but the orientation of one element is wrong. Hence, the breakpoint distance is* $bp(\pi) = 2$.

The breakpoint distance does not make assumptions on the evolutionary processes which rearranged the gene orders. This can be considered as an advantage (Blanchette et al., 1997). But the number of breakpoints is related to rearrangement distances (Nadeau and Taylor, 1984; Sankoff and Blanchette, 1998; Watterson et al., 1982). For example, an inversion creates at most two breakpoints and a transposition not more than three. Thus $\frac{bp(\pi)}{2}$, respectively $\frac{bp(\pi)}{3}$, are upper bounds for the inversion, respectively transposition, distance.

2.2.2 Common intervals

A common interval of a set of gene orders is a set of elements that appears consecutively in all gene orders. The order of the genes within such a cluster is not regarded and usually the strandedness of the genes is irrelevant for common intervals, too. Common intervals are a generalisation of conserved adjacencies. While conserved adjacencies consider only pairs, i.e. ordered element sets of size two, common intervals regard element sets of arbitrary size. Formally, a *common interval* of a set of permutations is defined as follows. Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of permutations of length n. A set of (unsigned) elements $c \subseteq \{1, \ldots, n\}$ is a common interval of Π if the elements of c appear in a consecutive interval in each of the k permutations. That is for each $\pi_i \in \Pi$, with $i \in [1 : k]$, there exist start and end indices s_i and e_i, with $1 \leq s_i \leq e_i \leq n$, such that $c = \{|\pi_i(j)| : j \in [s_i, e_i]\}$. A common interval can be specified by its set of elements or alternatively by the pair of start and end indices for each permutation. The set of common intervals of a set of permutations Π is denoted by $C(\Pi)$. The singletons $\{i\}$, $i \in \{1, 2, \ldots, n\}$, and the set $\{1, 2, \ldots, n\}$ of all elements are called *trivial* common intervals.

Example 2.2.2. *Consider the permutation* π = (7 5 -6 3 -1 -4 2). *The set of elements* $\{1, 2, 3, 4\}$ *is a common interval of* $\{\pi, \iota\}$ *because there is an interval in* ι *(starting at* 1 *and ending at* 4*) and in* π *(starting at position* 4 *and ending at* 7*) containing these elements. There are three more nontrivial common intervals. These are shown in Figure 2.2.*

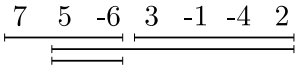


Figure 2.2 – The nontrivial common intervals of the signed permutation π in Example 2.2.2 and ι; each line represents one common interval containing the underlined elements

The value of common intervals for the comparison of gene orders was recognised in Heber and Stoye (2001a,b) (see also Heber et al. (2009)). But earlier, Uno and Yagiura (2000) presented an algorithm for computing the common intervals of two permutations in time $\mathcal{O}(n + K)$, where $K \leq \binom{n}{2}$ is the size of the output. The common intervals of a set of k permutations of size n can be computed in time $\mathcal{O}(kn + K)$ and $\mathcal{O}(n)$ space (Bergeron et al., 2005, 2008a; Heber et al., 2009; Heber and Stoye, 2001a,b). The key of these algorithms is the use of a generating subset of the common intervals, the so called irreducible intervals in Heber et al. (2009); Heber and Stoye (2001a,b) and strong common intervals in Bergeron et al. (2008a). Variations for the handling of multichromosomal and circular gene orders can be found in Heber et al. (2009); Heber and Stoye (2001a). Furthermore, a modified definition of common intervals incorporating the sign information of the elements is discussed in Heber and Stoye (2001a).

The generating sets of common intervals mentioned above are detailed in the following (see Bergeron et al. (2008a); Heber et al. (2009)). The main insight is that overlapping common intervals imply other common intervals where two common intervals c and c' are said to *overlap* iff

i) $c \cap c' \neq \emptyset$ and

ii) neither $c \subseteq c'$ nor $c' \subseteq c$.

If two common intervals c and c' overlap then $c \cap c'$, $c \setminus c'$, and $c' \setminus c$, $c \cup c'$ are also common intervals. A sequence $c_1, \ldots, c_l$, with $l \geq 1$, of common intervals is a *chain of common intervals* if every two successive intervals overlap. A chain that cannot be extended to its left or right is a *maximal* chain. A common interval c is called *reducible* if there exists a chain of common intervals $c_1, \ldots, c_l$ of length at least two such that $c = \cup_{i=1}^{l} c_i$ otherwise it is called *irreducible*. The set of irreducible intervals generates the set of common intervals. The number of the irreducible common intervals is smaller than n whereas the number of common intervals is only smaller than $\binom{n}{2}$. In any case the number of irreducible intervals is never larger than the number of common intervals. These observations are used in the algorithm for computing the common intervals presented in Heber et al. (2009); Heber and Stoye (2001a,b) which first constructs the irreducible common intervals in time $\mathcal{O}(kn)$. The set of common interval is generated in a second step from the irreducible common intervals in time

$\mathcal{O}(n+K)$. The set of irreducible intervals is one generating subset of common intervals which facilitate the design of efficient algorithms using common intervals.

Example 2.2.3. *Consider* $\pi = (7\ 5\ \text{-}6\ 3\ \text{-}1\ \text{-}4\ 2)$ *of Example 2.2.2. The common intervals* $\{1,\ldots,6\}$ *and* $\{5,6,7\}$ *do overlap because they do not have an empty intersection and neither is included in the other. The two intervals form a chain which is maximal. The two overlapping intervals generate the intervals* $\{5,6\}$ *(by intersection),* $\{1,\ldots,4\}$ *(set difference), and the trivial common intervals* $\{7\}$ *and* $\{1,\ldots,7\}$ *(set difference respectively union). All other pairs of common intervals do not overlap. For example,* $\{5,6\}$ *and* $\{5,6,7\}$ *have a nonempty intersection but* $\{5,6\}$ *is included in* $\{5,6,7\}$.

A general notion of generators of common intervals is introduced in Bergeron et al. (2008a) with an emphasis on a canonical generator that gives rise to the so called strong common intervals. The presented algorithms have the same asymptotic run time behaviour compared to the approach of Heber et al. (2009), but uses only very basic data structures. Strong intervals have been used for computing preserving rearrangement scenarios. These approaches are based on the so called strong interval tree data structure which represents the set of common intervals in a space efficient way (see Section 2.4 for definitions).

Finding the common intervals of permutations has applications in different areas. Common intervals can be used for the design of crossover operators for genetic algorithms solving permutation problems like the travelling salesman problem or single machine scheduling (see Uno and Yagiura (2000) and references therein). The application to modular decomposition of graphs is discussed in Bergeron et al. (2008a). Heber and Savage (2005) generalised common intervals to labelled trees. The parallel between computing the common intervals of permutations and the consecutive arrangement problem (Booth and Lueker, 1976) was recognised in Heber and Stoye (2001b). The consecutive arrangement problem (consecutive ones problem) is to find permutations of a set U in which certain given subsets $S \subseteq U$ occur consecutively. The problem of finding the common intervals of permutations reverses this problem.

2.2.3 Alternative gene cluster definitions

The consecutiveness property of common intervals may be too stringent for the identification of clusters of functionally related genes. Therefore, the concept of common intervals was generalised to allow for gaps in several ways (for an overview see Hoberman and Durand (2005)). One prominent example is given by *gene teams* (Bergeron et al., 2002a; Luc et al., 2003) that allow for gaps between successive elements of the gene cluster of size smaller than a given constant δ. Gene teams can be computed in polynomial time (Béal et al., 2004; Bergeron et al., 2002a). An alternative is to restricting the number of errors in the clusters, Böcker et al. (e.g. 2008). The compact representation of the gene

teams given by gene team trees of all possible values of δ presented in (Zhang and Leong, 2008) may support the crucial and difficult task to select the parameter δ in a meaningful way. Another approach, appealing because of its formal simplicity, was recently introduced by Zhu et al. (2009). This approach is to consider generalised gene adjacencies defined as pairs of genes located in a distance smaller than a given constant. Connected components of generalised gene adjacencies give rise to gene clusters representing a generalisation of gene teams.

2.3 Genome rearrangement scenarios and distances

Similarly as for the comparison of nucleotide and amino acid sequences, computing a shortest edit distance or the length of such a sequence is an important task also for gene arrangement data. For nucleotide and amino acid sequences a shortest, i.e. parsimonious, sequence of single (nucleotide or amino acid) insertions, deletions, and substitutions is searched which transform one sequence into another.

The two algorithmic problems, arising when gene arrangements are considered, are the *sorting problem* and the *distance problem*. Given two gene orders, the sorting problem asks for a shortest sequence of rearrangements transforming one gene order into the other. The distance problem asks for the length of such a shortest rearrangement sequence. These problems can be and have been studied in many different flavours. Usually, these problems are studied for a permutation π and the identity permutation. Of great importance for the algorithmical problems and the biological relevance of the results is the set of rearrangements considered for reconstructing the scenario. Often only a single type of rearrangement operations is considered. For example, if only inversions are allowed, the problems are usually called *sorting by inversions* and *inversion distance problem*. If more than one type of rearrangement operation is considered, one may assign weights to the different kinds of operations. The problem is then to find a sorting rearrangement sequence with a minimum sum of weights. Alternatively, to the type of the rearrangement certain properties of the rearrangements, e.g. their length, can be used to define weights. Forbidding certain rearrangements because of biologically motivated constraints is also a possibility which is extensively used in this work (Chapters 3 and 4), see also Section 2.4.

Properties of the analysed gene arrangements influence algorithmic considerations. Take for example the kind of genomes considered (circular or linear, uni- or multichromosomal), if the strandedness of the genes is known and relevant (signed or unsigned), or if the orientation of the gene arrangements is important (directed or undirected).

The following sections outline the developments and the current state of the art of genome rearrangement distance and genome rearrangement distance algorithmics. Sections 2.3.1 and 2.3.2 review the cases where only inversions, respectively only transpositions are con-

sidered. In Section 2.3.3 the state of the art for the tandem duplication random loss rearrangement is presented. The introduction of the algorithmics of the TDRL operation is detailed because Chapter 5 in this work gives new results for this model of rearrangement. An overview of the developments for approaches to genome rearrangement analyses considering multiple rearrangement operations is given in Section 2.3.4.

2.3.1 Inversions

Inversions are, at least from an algorithmic point of view, the best studied rearrangements. The initial work on sorting by inversions of Watterson et al. (1982) considered the case of unsigned circular permutations. Two simple heuristics have been presented. The "Ratchet algorithm" is a linear scan of the permutation and if position $\pi(i) \neq i$ then the inversion moving $\pi(i)$ to its position is applied. This heuristic sorts a permutation in at most $n-1$ steps, but it can perform arbitrary poorly, e.g. for $(n\ 1\ 2\ \ldots\ n-1)$ (Kececioglu and Sankoff, 1995). Furthermore, it was observed that an inversion can decrease the number of breakpoints by not more than two. This observation implies the now well known bound for the inversion distance $d(\pi) \geq \frac{bp(\pi)}{2}$. The second approach greedily applies inversions maximising the breakpoint reduction.

In Bafna and Pevzner (1996) the so called breakpoint graph for pairs of permutations was introduced (see also Kececioglu and Sankoff (1995)). This graph became one of the main theoretical tools for rearrangement analyses, in particular for sorting by inversions. A generalisation of the breakpoint graph to multiple gene orders is the basis of Caprara's algorithm for the inversion median problem (see Section 2.5.3). The breakpoint graph and its meaning for the sorting by inversions problem is discussed in more detail because of being helpful for understanding Caprara's algorithm and the extensions and modifications of Caprara's algorithm presented in this work.

The *breakpoint graph* of an unsigned permutation of size n is defined as an edge coloured graph with node set $\{0, \ldots, n+1\}$ where the adjacencies of π define the set of black edges and the adjacencies of the identity permutation define the set of grey edges. For the above definition the permutations are augmented by auxiliary elements 0 and $n+1$.

For the inversion distance $d(\pi) \geq bp(\pi) - cyc(\pi)$ holds, where $bp(\pi)$ is the number of breakpoints of π and $cyc(\pi)$ is the size of a decomposition of the edges of the breakpoint graph of π cycles alternating between edges of the two colours (Bafna and Pevzner, 1996). A decomposition into a maximum number of cycles gives the best bound. But finding such a decomposition has turned out to be the obstacle for efficient algorithms sorting unsigned permutations by inversions. The problem was shown to be NP-hard in Caprara (1999).

The lessons learned from the unsigned case have been applied to the signed case in Bafna and Pevzner (1996); Kececioglu and Sankoff (1994) to design approximation algo-

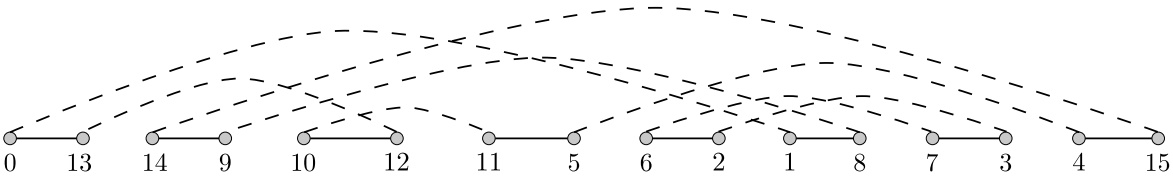


Figure 2.3 – Breakpoint graph for the signed permutation in Example 2.3.1; black edges (horizontal lines) show represent the adjacencies of the permutation π and grey lines the adjacencies of the identity permutation

rithms. Most importantly a "tremendous simplification" with respect to the maximum alternating cycle decomposition of the breakpoint graph was observed when defined for signed permutations (Kececioglu and Sankoff, 1994). Let π' be the unsigned permutation of size $2n$ generated from a signed permutation π of size n by replacing for all $i \in [1:n]$

- a positive element $+i$ by the pair of elements $2i-1$ and $2i$ and
- a negative element $-i$ by the pair of elements $2i$ and $2i-1$.

Note, inversions for π can be mimicked by inversions for π' and the unsigned permutation generated from the signed identity permutation is the identity permutation. Hence, sorting by inversions can be described on the transformed unsigned permutations. Furthermore, the vertices $2i-1$ and $2i$ are connected by a black and a grey edge, i.e. each such pair defines an alternating cycle. The *breakpoint graph* of a signed permutation is obtained by removing those $2n$ cycles. Every vertex in the breakpoint graph is incident to one black and one grey edge, i.e. every vertex has degree two. Thus, the breakpoint graph of a signed permutation can be trivially decomposed into a maximum number of alternating cycles. This enabled the computation of the lower bound $d(\pi) \geq bp(\pi) - cyc(\pi)$ in linear time. In experiments this bound was shown to be extremely tight.

Example 2.3.1. *Figure 2.3 shows the breakpoint graph for the signed permutation* $\pi =$ (7 5 -6 3 -1 -4 2)*. The breakpoint graph has two cycles. One including the nodes* 2*,* 3*,* 7*, and* 8 *the other including the remaining elements. The permutation framed by the elements* 0 *and* 8 *has eight breakpoints. Thus, the permutation can not be sorted to the identity with less than six inversions.*

Hannenhalli and Pevzner (1995b) gave the first polynomial time ($\mathcal{O}(n^4)$) algorithm for sorting signed permutations. This breakthrough was also based on the breakpoint graph as introduced above. The main contribution was the identification of the missing parameters (called hurdles, super hurdles, and fortresses) separating the bound from the distance. These parameters can be determined from certain configurations of connected components in the breakpoint graph. Subsequently more efficient algorithms for the inversion distance problem

have been devised: $\mathcal{O}(n^2\alpha(n))$ where α is the inverse Ackermann function (Berman and Hannenhalli, 1996), $\mathcal{O}(n^2)$ (Kaplan et al., 1999), and finally $\mathcal{O}(n)$ (Bader et al., 2001). The formal aspects of the sorting by inversions and the inversion distance problem have been significantly simplified in Bergeron et al. (2002b, 2004b). For precise details of the sorting by inversion theory the reader is referred to (Pevzner, 2000, chapter 10) or Bergeron et al. (2004b). While the inversion distance can be computed in linear time, the currently best result for sorting by inversions can be found in Tannier et al. (2007) where an algorithm with run time $\mathcal{O}(n^{3/2}\sqrt{\log n})$ was presented.

Example 2.3.2. *The lower bound for the permutation of Example 2.3.1 turns out to be identical to the inversion distance. That is it is possible to transform π into the identity permutation with six inversions. One such sorting scenario (computed with the online MGR program (*Bourque and Pevzner*, 2002)) is shown below. The inverted elements are underlined.*

$$
\begin{aligned}
\pi = (7 \quad & \underline{5} \quad -6 \quad 3 \quad -1 \quad -4 \quad 2) \\
(7 \quad & \underline{-5} \quad -6 \quad 3 \quad -1 \quad -4 \quad 2) \\
(5 \quad & \underline{-7 \quad -6} \quad 3 \quad -1 \quad -4 \quad 2) \\
(5 \quad & 6 \quad 7 \quad 3 \quad -1 \quad -4 \quad 2) \\
(1 \quad & -3 \quad -7 \quad -6 \quad -5 \quad -4 \quad 2) \\
(1 \quad & \underline{-3 \quad -2} \quad 4 \quad 5 \quad 6 \quad 7) \\
\iota = (1 \quad & 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7)
\end{aligned}
$$

While a single sorting sequence of inversions can be computed easily, it may also be of interest to know the set of all sorting sequences, e.g. when hypotheses about rearrangement scenarios have to be tested or for estimating the significance of a reconstructed scenario.

As a subtask all sorting inversions have to be determined, i.e. inversions reducing the distance by one. A straightforward algorithm enumerates all $\binom{n}{2}$ inversions and computes the inversion distance for each resulting permutation. This yields a run time of $\Theta(n^3)$. A first successful attempt to improve this was presented in Ajana et al. (2002) where a method was given to compute all sorting inversions heuristically. This was used to explore all sorting scenarios in order to test if inversions are symmetric around the replication origins of bacterial genomes. An exact algorithm for the enumeration of all sorting inversions with run time $\mathcal{O}(n^3)$ was presented in Siepel (2003). An approach for enumerating nearly all sorting and distance neutral inversions with run time $\mathcal{O}(n^2)$ was presented in Bernt et al. (2006a). The problem of enumerating the potentially extremely large set of all sorting sequences of inversions is addressed in Braga et al. (2008).

2.3.2 Transpositions

The computational complexity of sorting by transpositions and the transposition distance problem is an open problem. Currently no efficient algorithm exists for computing the distance of a sorting scenario. In Bafna and Pevzner (1995) lower and upper bound for the transposition distance and 1.5-approximation algorithm with run time $\mathcal{O}(n^2)$ have been presented. Later on these results have been simplified and the run time was reduced to $\mathcal{O}(n^{3/2}\sqrt{\log n})$ by using tree data structures (Hartman, 2003; Hartman and Shamir, 2006). A further improvement of the run time to $\mathcal{O}(n \log n)$ was reported recently (Feng and Zhu, 2007). Currently, the best approximation algorithms have an approximation factor of 1.375 and the run time is in $\mathcal{O}(n^2)$ (Elias and Hartman, 2006; Labarre, 2006).

2.3.3 Tandem duplication random loss

A *tandem duplication random loss* (TDRL) operation duplicates a continuous segment of genes in tandem followed by the loss of one copy of each of the redundant genes. This operation is especially important for the study of mitochondrial gene orders because the gene content of mitochondrial genomes tends to be preserved during evolution (Boore, 1999), i.e. complete loss can be assumed.

The TDRL rearrangement operation was studied initially by Chaudhuri et al. (2006). The authors proposed the cost function α^l for weighting a single TDRL duplicating l genes, where $\alpha \geq 1$ is a constant. Results for the problems of finding a minimal sorting scenario of TDRLs, the length of such a scenario, median problems, and properties of TDRLs have been presented for the cases $\alpha = 1$ and $\alpha \geq 2$. Because of the relevance for the results presented in Chapter 5 especially the results for $\alpha = 1$ are discussed in more detail in the following.

A variant of the TDRL model is studied in Bouvel and Rossin (2009) limiting the number l of elements duplicated by a TDRL by using a modified cost function, i.e. where the cost of a TDRL of length l is 1 if $l \leq K$ and ∞ otherwise, for a constant parameter $K \in \{1, \ldots, n\} \cup \{\infty\}$.

For the case of $\alpha \geq 2$ it is sufficient to consider TDRLs duplicating intervals of size two (Chaudhuri et al., 2006). Hence, the TDRL distance for $\alpha \geq 2$ is equal to the "bubble sort distance" given by the number of inversions in a permutation, i.e. $|\{(i, j) : i > j, \pi(i) < \pi(j)\}|$. This distance can be computed in time $\mathcal{O}(n \log n)$. The median problem for this distance was studied in different contexts and is shown to be NP-hard for more than three permutations (Dwork et al., 2001).

Most interesting, for the computational and combinatorial results presented in Chapter 5, is the case $\alpha = 1$. In this cost model the length of the tandem duplicated interval has no influence on the cost of a TDRL, hence if $\alpha = 1$, whole genome duplications can be assumed.

Chaudhuri et al. (2006) recognised the equivalence of TDRLs and steps of the classic, i.e. least significant digit, radix sort algorithm.

The presented algorithm for computing a TDRL scenario of minimal length and the TDRL distance is based on the notion of *maximal increasing substrings* of a permutation, which are defined as follows. Let $\pi = (\pi(1) \ \dots \ \pi(n))$ be an unsigned permutation of length n. Note, for the TDRL model of genome rearrangement only unsigned permutations are considered. A substring $(\pi(i) \ \dots \ \pi(j))$, with $1 \leq i \leq j \leq n$, is *increasing* if $\pi(k) < \pi(k+1)$ holds for all $k \in [i : j-1]$ and it is *maximal increasing* if it cannot be extended to the left or right. Let $\varrho(\pi)$ denote the number of maximal increasing substrings of permutation π. The algorithm for sorting the identity permutation to a given permutation π starts with labelling the elements of the i-th maximal increasing substring of π with the binary representation of i. The identity permutation is transformed into π by applying the radix sort algorithm to the binary representation of the element's maximal increasing substring indices. That is in the k-th step of the algorithm a whole genome TDRL is applied which keeps all the elements in the first (respectively second) copy which have a 0 (respectively 1) at the k-th least significant digit in the binary representation of the element's maximal increasing substring index. More formally, a TDRL applied to the n elements of a permutation is specified by a binary string of length n where

- a 0 at the e-th position indicates that element e is kept in the first copy (i.e. deleted in the second copy) and
- a 1 at the e-th position indicates that the element e is kept in the second copy (i.e. deleted in the first copy).

The e-th position of the binary string corresponding to the TDRL applied in the k-th step of the algorithm is given by the k-th least significant digit of the binary representation of the index of the maximum increasing substring that contains element e. After $\lceil \log \varrho(\pi) \rceil$ steps of the algorithm the identity is transformed into the target permutation π. Chaudhuri et al. (2006) have proven that sequence of TDRLs computed by this algorithm is minimal. Thus, the TDRL distance is given by $d(\iota, \pi) = \lceil \log \varrho(\pi) \rceil$ and can be computed in time $\mathcal{O}(n)$.

Example 2.3.3. *The permutation* $\pi = (4\ 2\ 6\ 3\ 5\ 1)$ *has four maximal increasing substrings,* (4), $(2\ 6)$, $(3\ 5)$, *and* (1). *That is* $\varrho(\pi) = 4$ *and consequently the TDRL distance is two. The TDRL scenario transforming* ι *into* π *is constructed as follows (see Figure 2.4). The first TDRL keeps the elements in the first copy which have a* 0 *in the least significant digit, i.e. the elements* 3, 4, *and* 5 *belonging to the maximal increasing substrings* 0 *and* 2. *The remaining elements have a* 1 *at the least significant digit and are therefore kept in the second copy. Similarly, in the second TDRL the elements of the first and second maximal increasing substrings are kept in the first copy of the TDRL because the binary representations of* 0 *and*

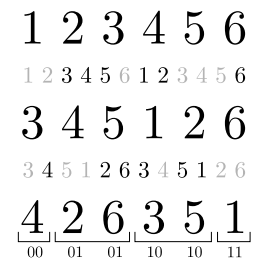


Figure 2.4 – A sorting TDRL scenario for ι to $\pi = (4\ 2\ 6\ 3\ 5\ 1)$ constructed with the method from Chaudhuri et al. (2006); the four increasing substrings of π are indicated on the bottom together with their binary representation; the two TDRLs are shown with the intermediate whole genome duplication product where grey shaded elements indicted lost genes

2 *have a* 0 *at the second least significant digit. The remaining elements are kept in the second copy.*

TDRLs are strongly asymmetric (Chaudhuri et al., 2006), i.e. mostly the effects of a TDRL cannot be reversed with a single TDRL. More precisely, only TDRLs with a binary representation of the form $0^*1^*0^*1^*$ are symmetric. Where the binary representation is given with respect to the elements of the identity permutation, x^+ denotes at least one repetition of symbol x, and x^* zero or more repetitions of symbol x. Symmetric TDRLs are TDRLs without an effect on the order of the elements (corresponding to 0^*1^*) and transpositions ($0^*1^+0^+1^*$). The number of those symmetric TDRLs is $\mathcal{O}(n^3)$ and therefore negligible compared to the number of 2^n possible TDRLs. Thus, the probability of an asymmetric step is exponential in the length of the gene order, when in the evolutionary model the loss of genes is assumed to be uniformly at random. Based on this findings a directed median problem was defined which is discussed briefly in Section 2.5.4.

The TDRL model of genome rearrangement is applied and studied in more detail in this work. The Algorithms `CREx` and `TreeREx` (Chapter 4) for reconstructing pairwise scenarios respectively of rearrangements in phylogenetic trees include TDRLs in the set of considered rearrangement operations. Additionally, an application of the asymmetry for the phylogenetic reconstruction of gene order events is presented there. The question whether the TDRL scenario is unique is studied in Chapter 5.

2.3.4 Mixed rearrangements

The assumption that only one kind of rearrangements has occurred during gene order evolution is most likely unrealistic (see Section 2.1 and in Sections 4.4 and 4.5 for many examples of different rearrangements in mitochondrial gene orders). For realistic reconstructions of the

rearrangement history of given gene orders different types rearrangement operations should be considered.

For unichromosomal genomes, e.g. mitochondrial genomes, a combination of the presented rearrangement operations can be considered as sufficient (Boore, 1999). This is inversions, transpositions, and tandem duplication random loss. Additionally inverse transpositions may be considered. But it is unclear how these operations should be weighted. Usually more weight is given to transpositions than to inversions (Bader et al., 2008; Blanchette et al., 1996; Eriksen, 2003, 2002).

Some algorithmic approaches are available for the construction of combined inversions and transposition rearrangement scenarios. Among the first is DERANGE (Sankoff et al., 1992; Sankoff, 1992), a branch and bound algorithm searching for a shortest scenario consisting of transpositions and inversions for two unsigned gene orders. In this early work the possibility of weighting the operations differently was already explored and analysed. With DERANGE II (Blanchette et al., 1996) the possibility to handle signed permutations was added. The currently best known results are polynomial time approximation algorithms that allow for the consideration of inversions, transpositions and inverse transpositions in a weighted fashion (see Bader and Ohlebusch (2007) and references therein).

For the study of multichromosomal genomes even more rearrangement types have to be considered. These are the *fusion* of two chromosomes into one chromosome and the inverse operation *fission* that splits one chromosome into two parts. Additionally, the exchange of genetic material between chromosomes (suffixes or prefixes) is called *translocation*. In their seminal work Hannenhalli and Pevzner (1995a) presented a polynomial algorithm to compute a shortest sequence of rearrangements for two multichromosomal gene orders using inversions, fissions, fusions, and translocations.

Despite, combined distance measures including transpositions are rare, recently some very promising approaches for this problem have been suggested (Bergeron et al., 2006, 2008b; Yancopoulos et al., 2005). These are based on the so called *double cut and join* (DCJ) operation, which cuts a (multichromosomal) gene order at two positions and rejoins the resulting fragments. Translocations, inversions, creation of circular intermediate chromosomes and their reintegration into a chromosome can be produced with this operation. The subsequent generation of a circular intermediate and its reintegration is equivalent to a block interchange. While a transposition exchanges two adjacent intervals, the adjacency restriction is removed for *block interchanges*, i.e. two intervals are exchanged that do not have to be adjacent. The underlying theory resembles the Hannenhalli-Pevzner-Theory of sorting by inversions, i.e. the distance can be computed by counting the cycles in the breakpoint graph without the rather complicated correction terms. The additional possibility of allowing insertions, deletions and duplications was studied in Bader (2009); Yancopoulos and Friedberg (2008). Closed

formulas for the number of sorting DCJ operations have recently been derived (Braga and Stoye, 2009; Ouangraoua and Bergeron, 2009).

The considered gene arrangements may contain duplicated or deleted genes. Then the approaches developed for permutations can not be applied. The so called *exemplar approach* is one method to handle this problem (Sankoff, 1999). This is to choose a permutation minimising some criteria, e.g. a rearrangement distance, from the permutations derived by trying all possibilities to assign unique names to the duplicated genes. Some approaches exist also for insertions and deletions. In El-Mabrouk (2000b), the Hannenhalli Pevzner theory for sorting by inversions is extended to allow insertions and deletions, later on the possibility to handle duplications has been added (Marron et al., 2003).

2.4 Constrained rearrangement analyses

The study of shortest rearrangement scenarios is useful for the reconstruction of phylogenetic information, e.g. rearrangements distances and scenarios. The approaches for solving the distance and sorting problems arising from the gene arrangement data have made a fascinating progress during the last decades.

An approach for increasing the reliability of rearrangement methods is to consider additional biologically motivated constraints. An overview of the approaches considering biological constraints for rearrangement analyses is given in this section, with special emphasis is on the sorting by preserving inversions problem, in particular, the work of Bérard et al. (2007) which inspired the results presented in Chapter 3 in this work.

2.4.1 Problem definition

The problem of sorting by (common interval) preserving rearrangements is defined as follows. Let $C(\Pi)$ denote the common intervals of a set of gene orders Π. Given two gene orders $\Pi = \{\pi, \sigma\}$, a rearrangement ρ for π *preserves* the common intervals of Π if $C(\Pi) = C(\Pi \cup \{\pi \circ \rho\})$ (according to Bergeron and Stoye (2006)). Alternatively, ρ is said to be *preserving* with respect to (the common intervals) of Π. If the context is clear, ρ is just called preserving. Two sets X and Y commute iff $X \subset Y$, $Y \subset X$, or $X \cap Y = \emptyset$. An inversion ρ is preserving for a common interval c iff ρ and c *commute* (Bérard et al., 2004). Recall, inversions and common intervals are defined as sets. Commuting sets (intervals and inversions) are an important concept for constrained rearrangement analyses (see below). The definition of preserving inversions can be extended to rearrangement scenarios. A rearrangement scenario $\rho_1, \ldots, \rho_l$, with $l \geq 1$, *preserves* the common intervals of Π if for all $i \in [1 : l]$, $C(\Pi) = C(\Pi \cup \{\pi_1 \circ \rho_1 \circ \ldots \circ \rho_i\})$ holds. A shortest preserving rearrangement scenario is called *sorting* (respectively *parsimonious*). The length of such a shortest scenario is called *preserving*

rearrangement distance. In the literature the term *perfect* is used instead of preserving sometimes.

Example 2.4.1. *The sorting inversion scenario given in Example 2.3.2 is not preserving. The second inversion* $\{5, 7\}$ *destroys the common intervals* $\{5, 6\}$ *and* $\{1, \ldots, 6\}$ *which are recovered with the third inversion. In the fourth step of the inversion sequence the common interval* $\{1, \ldots, 4\}$ *is destroyed.*

This definition can easily be modified to account for different combinatorial structures to be preserved during the sorting, e.g. conserved intervals or gene teams. Similar as for the unrestricted case, different sets of considered rearrangement operations may be considered. The case of sorting by preserving inversions is reviewed in the following. There will be a final discussion on different rearrangement types and constraints that have been considered in the literature.

2.4.2 Common interval preserving inversions

Bergeron et al. (2004a) is one of the first works on rearrangement scenarios that preserve some combinatorial structures, where conserved intervals have been considered. A *conserved interval* is a common intervals with the property that it is flanked by the same pair of elements in the same or opposite direction. The preserving property for inversions and transposition types was studied for this set of intervals. This subset of the common intervals is of special interest because there is a relationship between the problem of sorting by inversions and conserved intervals (see also Bergeron et al. (2002b)).

The first study considering the set of common interval for restricting the set of allowed inversion was presented in Figeac and Varré (2004) where the NP-completeness of the problem was established. Moreover, exact algorithms for the sorting problem have been devised. These algorithms have exponential worst case run time behaviour. A basic operation used in the algorithm is to collapse a common interval in a permutation. This operation replaces the elements of the common interval in the input permutations by a single element and renames the remaining elements in an order and orientation maintaining manner such that the result is again a permutation. In a first step, sets of common intervals have been considered which are pairwise not overlapping, i.e. they are pairwise disjoint or one is included in the other. So to say sets of pairwise commuting common intervals are considered. The idea of the algorithm is to sort the common intervals of the permutations separately where smaller common intervals are processed before larger ones. But there are two possibilities, sorting the elements of the interval to the increasing or to the decreasing order. A brute force approach tests both possibilities for all common intervals. This is improved by Figeac and Varré (2004) observing that the decision can be made locally if one of the possibilities needs less inversions than the other. For that case the interval is collapsed to a single positive

or negative value depending on the smaller value. Otherwise, both possibilities have to be tested again. This leads to a run time of $\mathcal{O}(2^{K'}n)$, where n is the size of the permutation and K' the number of intervals with a tied decision for sorting to the increasing or decreasing order. If K' is in $\mathcal{O}(\log n)$, the algorithm is polynomial. In a second step all common intervals, including overlapping ones, are regarded. Given a chain of pairwise overlapping common intervals $(c_1, \ldots, c_l)$ a *multi-block* is a list of consecutive intervals $(b_i, \ldots, b_j)$ such that $|\max(b_l)| + 1 = |\min(b_{l+1})|$ or $|\min(b_l)| - 1 = |\max(b_{l+1})|$ for $i \leq l < j$. In the former case the multi-block is called increasing, else decreasing. Preserving sorting is achieved by sorting each interval b_l separately to the increasing or decreasing order if one of the two options needs less inversions. Otherwise, if none of the two options is smaller, b_l is sorted to the same order as the multi-block. Note, in any case the decision is determined. The blocks sorted to increasing (decreasing) order in an increasing (decreasing) multi-block are inverted afterwards. Non overlapping intervals are treated as described above. This leads to a run time of $\mathcal{O}(2^{K'}n + Kn)$ for a permutation of size n with K common interval and K' as defined above.

Sorting inversions scenarios have been analysed with respect to their "preservedness", measured as the number of inversions in the scenario which commute with all common intervals (Bérard et al., 2004). Furthermore, the relation of commuting inversion scenarios, i.e. scenarios consisting of pairwise commuting inversions, and shortest inversions scenarios has been studied. It was shown that a commuting inversion scenario is preserving if it is minimal with respect to the inversion distance. The general question, about the existence of a polynomial time algorithm deciding whether there is a minimum inversion scenario that is preserving was answered positively in Diekmann et al. (2007); Sagot and Tannier (2005).

The use of the signed strong interval tree data structure — an extension of the PQ tree data structure (Booth and Lueker, 1976) — was shown in Bérard et al. (2007, 2008b) to be beneficial for solving the problem of computing sorting scenarios of preserving inversions efficiently. Most importantly, the problem was shown to be polynomial time solvable for a large class of permutations and fixed parameter tractable (Downey and Fellows, 1999) for the remaining permutations. The following description of sorting by preserving inversions with the signed strong interval trees is based on Bérard et al. (2007, 2008b), the reader is referred there for the formal proofs.

The foundation of the algorithms is the subset of pairwise commuting common intervals. A common interval is called a *strong* common interval if it does not overlap with any other common interval. That is two strong common intervals either are disjoint or one is included in the other. Hence, the strong common intervals of a set of permutations Π can be represented as a tree, called *strong interval tree*, where the nodes correspond to the strong common intervals and the edges are defined by the minimal inclusion relation. That is two strong

common intervals c and c' are connected by an edge if $c \subset c'$ and there is no strong common interval c'' with $c \subset c'' \subset c'$. The strong interval tree of a set of permutations Π is denoted by $T(\Pi)$. This data structure was introduced to genome rearrangement problems in Bérard et al. (2007); Parida (2006).

Strong interval trees for sets of two permutations are regarded in the following where, without loss of generality, one permutation is the identity permutation. Let I be a node of the strong interval tree $T(\{\pi, \iota\})$, $T(\pi)$ for short, with l child nodes $\{I_1, \ldots, I_l\}$. Then the l child nodes of I define a partition of I into maximal strong intervals. The *quotient permutation* $\pi_{|I}$ of I with respect to π is defined as the (unsigned) permutation of $\{1, \ldots, l\}$ where i precedes j in $\pi_{|I}$ if any element of I_i is smaller than any element of I_j, with $i, j \in [1 : l]$. That is I_i precedes I_j in the identity permutation.

A quotient permutation of a node in $T(\pi)$ is either

1. *increasing linear*, i.e. $(1 \ \ldots \ l)$,
2. *decreasing linear*, i.e. $(l \ \ldots \ 1)$,
3. or *prime*.

An essential observation is that the quotient permutation "inherits" the common intervals from the permutation π in the following sense. A subset of $P = \{1, \ldots, l\}$ is a common interval in $\pi_{|I}$ iff $\bigcup_{i \in P} I_i$ is a common interval of π. Moreover (see Bérard et al., 2007, Theorem 1), $J \subseteq \{1, \ldots, n\}$ is a common interval of π (and ι) iff either

1. J is a node of $T(\pi)$ or
2. J is the union of consecutive children of a linear node.

Hence, strong interval trees represent the set of all common intervals and consequently sorting with preserving inversions can be described in terms of the signed strong interval tree. That is an inversion scenario is preserving iff each inversion is a node of the strong interval tree or the union of children of a prime node in the strong interval tree. An inversion is a node (a union of nodes) of the strong interval tree if it acts exactly on the elements of the corresponding strong interval.

The set of strong common intervals and strong interval trees can be computed in linear time (Bergeron et al., 2008a). Remark, the number of nodes of a strong interval tree is in $\mathcal{O}(n)$. Thus, it is an efficient data structure for storing the set of common intervals of two permutations and allows for efficient identification of preserving inversions.

The question for identifying the sorting preserving inversions among the preserving inversions is solved with the signed strong interval tree data structure. A *signed strong interval tree* is the strong interval tree where an additional sign $+$ or $-$ is assigned to the nodes in the following way:

1. leaf nodes are signed with the sign of the corresponding element in π,
2. linear nodes are signed with a $+$ (respectively $-$) if the corresponding quotient permutation is increasing (respectively decreasing) linear, and

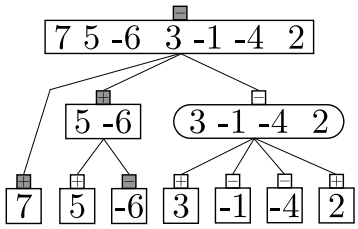


Figure 2.5 – Signed strong interval tree for the permutation of Example 2.4.2; linear nodes are depicted as rectangles and prime nodes as round nodes; the signs are given on top of each node; grey shaded sign indicate a sign difference to the parent node

3. prime nodes with a linear parent node inherit the sign from the parent iff it is linear.

Remark that some nodes, i.e. prime nodes with prime node parents, remain unsigned. Signed strong interval trees having a sign assigned to every node are called *unambiguous*, else *ambiguous*. Each child node of a node I in a strong interval tree corresponds to an element of the quotient permutation $\pi_{|I}$. Assigning the signs of the nodes in the signed strong interval tree to the corresponding elements defines the *signed quotient permutation*. Note, if the sign of a child node is unknown, the quotient permutation is *partially signed*. Elements of quotient permutations corresponding to nodes without a sign are marked with $\pm$.

Example 2.4.2. *Consider* $\pi = (7\ 5\ {-6}\ 3\ {-1}\ {-4}\ 2)$ *used in previous examples. The permutation has two nontrivial strong common intervals* $\{5, 6\}$ *and* $\{1, \ldots, 4\}$*. The other two common intervals do overlap. The strong interval tree of the permutation is shown in Figure 2.5. The node, corresponding to the strong common interval* $I = \{1, \ldots, 4\}$*, has four child nodes corresponding to the trivial strong common intervals* $\{1\}, \ldots, \{4\}$*, which define a partition of* I *into maximal strong common intervals. The signed quotient permutation* $\pi_{|I}$ *is given by* $(3\ 1\ 4\ 2)$*. The quotient permutation is prime and therefore also the node. The child nodes of the node* I *are all leaves, i.e. they are signed according to the sign of the corresponding elements. Thus, the signed quotient permutation is* $(3\ {-1}\ {-4}\ 2)$*.*

The root node of the strong interval tree has three child nodes $\{1, \ldots, 4\}$*,* $\{5, 6\}$*, and* $\{7\}$*. The quotient permutation of the root node is given by* $(3\ 2\ 1)$*. Thus, the quotient permutation as well as the root node are decreasing linear. The signed quotient permutation is given by* $\pi_{|J} = (3\ 2\ {-1})$*.*

Given all these necessary definitions, the computation of a sorting preserving inversion scenario is guided by the signed strong interval tree. In particular, the inversion of a node I belongs to any sorting preserving scenario iff I has a linear parent with a different sign (Bérard et al., 2007, Lemma 2). Hence, if every node in $T(\pi)$ is linear, the sorting preserving inversion scenario is given by the inversions of the nodes with a sign different to the sign of their parent. The computation of a sorting parsimonious scenario can be achieved in $\mathcal{O}(n)$ time by a single traversal of the nodes of $T(\pi)$.

Recall, no union of a subset of a prime node's children is a common interval. Thus, the order of the intervals corresponding the children of a prime node can be rearranged without destroying a common interval. The inversions of a sorting scenario for the quotient permutation of a prime node imply preserving inversions for the permutation itself. The crucial question is whether the sorting of the quotient permutation has to be to the identity ι or to the inverted identity permutation $-\iota$. If a prime node has a linear parent, this decision is determined by the sign inherited from the linear parent. This is sorting to ι (respectively $-\iota$) when the prime node has sign + (respectively −) results in a parsimonious preserving inversion scenario (Bérard et al., 2007, Theorem 3). Thus, for unambiguous signed strong interval trees the worst case run time of the algorithm is limited by the run time of the fastest algorithm for sorting by inversions, i.e. currently $\mathcal{O}(n^{3/2}\sqrt{\log n})$ (Tannier et al., 2007).

For ambiguous trees it is unknown if the quotient permutations of prime nodes with prime parent node have to be sorted to ι or $-\iota$ because such nodes have no sign. Furthermore, the sign of the elements in quotient permutations corresponding to prime child nodes is unknown. A simple brute force method tries all possible sign assignments to the prime nodes with prime node parent and applies the methods for unambiguous signed strong interval trees. The assignment resulting in minimal number of inversions implies a sorting preserving inversion scenario. This algorithm has run time $\mathcal{O}(2^p n\sqrt{n \log n})$, where p is the number of prime nodes that have no linear parent.

Example 2.4.3. *Given this, a sorting by preserving inversions scenario for the permutation in Example 2.4.2 is computed as follows. A sorting inversion scenario for the quotient permutation to the inverse of the identity is computed. This is because the prime node inherited a negative sign from its linear parent. A sorting inversion scenario is given below on the left and the corresponding inversions of a sorting preserving inversion scenario on the right.*

$$
\begin{array}{rcccc}
\pi_{|I} = (& 3 & \underline{\text{-1}} & \underline{\text{-4}} & 2) \\
(& \underline{3} & \underline{4} & 1 & 2) \\
(& \text{-4} & \text{-3} & \underline{1} & \underline{2}) \\
(& \text{-4} & \text{-3} & \text{-2} & \text{-1})
\end{array}
\qquad\qquad
\begin{array}{rccccccc}
\pi = (& 7 & 5 & \text{-6} & 3 & \underline{\text{-1}} & \underline{\text{-4}} & 2) \\
(& 7 & 5 & \text{-6} & \underline{3} & \underline{4} & 1 & 2) \\
(& 7 & 5 & \text{-6} & \text{-4} & \text{-3} & \underline{1} & \underline{2}) \\
(& 7 & 5 & \text{-6} & \text{-4} & \text{-3} & \text{-2} & \text{-1})
\end{array}
$$

The remaining inversions of the sorting preserving inversion scenario for π correspond to nodes with a different sign as its parent node.

$$
\begin{array}{rccccccc}
 & (\ 7 & 5 & \underline{-6} & -4 & -3 & -2 & -1) \\
 & (\ \underline{7} & 5 & 6 & -4 & -3 & -2 & -1) \\
 & (-7 & \underline{5} & \underline{6} & -4 & -3 & -2 & -1) \\
 & (-7 & -6 & -5 & -4 & -3 & -2 & -1) \\
\iota = & (\ 1 & 2 & 3 & 4 & 5 & 6 & 7)
\end{array}
$$

The preserving inversion distance is seven. Recall, the inversion distance is six. Thus, there is no sorting inversion scenario which is also preserving for the permutation π.

This approach was improved in Bérard et al. (2008b). The key is that sorting inversion scenarios for the signed quotient permutation of a prime node may have different lengths if sorted to the identity or negative identity. For the formal description some additional definitions are necessary. A *completion* of a partially signed permutation π is the signed permutation π' obtained from π by giving signs to all unsigned elements of π. For a permutation π let $d^+(\pi)$ (respectively $d^-(\pi)$) denote the minimum inversion distance to ι (respectively -ι) over all completions of a partially signed permutation π. Furthermore, π^+ (respectively π^-) denotes a completion that realises the minimum inversion distance. Then a partially signed permutation π is called:

1. negative if $d^-(\pi) < d^+(\pi)$,
2. positive if $d^+(\pi) < d^-(\pi)$, and
3. neutral otherwise (if $d^+(\pi) = d^-(\pi)$).

This definition applies to partially signed quotient permutations of prime nodes with prime child nodes. Note, a similar definition was also presented earlier in the approach of Figeac and Varré (2004). The method of Bérard et al. (2008b) computes for each prime node I, occurring in a post order traversal of $T(\pi)$, the values $d^+(\pi_{|I})$ and $d^+(\pi_{|I})$. If $\pi_{|I}$ is neutral, nothing is done, but the already calculated permutations π^+ and π^- are stored. Otherwise, if $\pi_{|I}$ is not neutral, the smaller alternative is chosen and the decision is propagated to the prime nodes in the subtree, i.e. if the sign of a node is $+$ $(-)$ then π^+ (π^-) is used for defining the signs of the prime nodes in the subtree. The run time of the algorithm is exponential in the maximum number prime child nodes for a prime node in the strong interval tree, denoted by p'. That is the run time is $\mathcal{O}(2^{p'} n\sqrt{n \log n})$ which is a great improvement compared to the brute force algorithm.

Yet without this improvement the average time complexity of the algorithm from Bérard et al. (2007) for computing a parsimonious preserving scenario is polynomial, bounded by the run time of the best known algorithm for sorting by inversions (Bouvel et al., 2009), i.e. currently $\mathcal{O}(n^{3/2}\sqrt{\log n})$. Besides this, the study presents results on the average length of sorting preserving inversion scenarios and gives a connection to another tree data structure, namely Schröder trees.

2.4.3 Other considered rearrangements

Parsimonious preserving double cut and join (DCJ) rearrangement scenarios have been studied recently in Bérard et al. (2008a, 2009). The general problem is NP-hard as for sorting with preserving inversions, but for certain families of common intervals contrasting complexity results have been shown. That is if the common intervals are nested (i.e. no interval overlaps another interval), the preserving sorting problem is NP-hard for inversions but polynomial time solvable for DCJ rearrangements. For weakly separable sets of common intervals (i.e. every interval overlaps with at least another interval or is the union of intervals) the opposite is the case, i.e. the problem is linear time solvable when inversions are considered but NP-hard for DCJ.

The problem of finding preserving inversion and transposition scenarios for unsigned gene orders of the same species was studied in Parida (2006). The main theme of Bérard et al. (2007); Parida (2006), i.e. the reconstruction of rearrangements by identifying patterns in the strong interval tree, is also applied and extended in the `CREx` method presented in Chapter 4 for computing pairwise rearrangement scenarios for signed gene orders utilising the four types of rearrangement operations relevant for mitochondrial gene orders.

Ideas for handling of deleted and duplicated genes with strong interval trees are described in Landau et al. (2005).

2.4.4 Alternative considered constraints

Inversions tend to be symmetric around the origin and terminus of replication in circular bacterial chromosomes (Eisen et al., 2000; Tillier and Collins, 2000). This was studied through the enumeration of sorting sequences of inversions in Ajana et al. (2002). The sorting by inversions problem constrained to inversion that are symmetric around the origin and terminus of replication was studied in Ohlebusch et al. (2007).

The length of the rearrangements is also a parameter which can be constrained. This is of interest because the length or the rearrangements may be of a certain distribution (Sankoff et al., 2004), e.g. favouring smaller or large scale rearrangements. Thus, considering length weighted versions of the rearrangement problems is promising, too. Bounds and approximation algorithms for a length weighted variant of sorting by inversions was studied in Bender et al. (2008) where a cost function l^α, with $\alpha \geq 0$, is used for weighting an inversion acting on l genes.

In prokaryotic and mitochondrial genomes, rearrangements tend to occur more frequently in the proximity of the origin and terminus of replication (Fonseca and Harris, 2008; Suyama and Bork, 2001). If there are fragile regions in the genomes of higher organisms has been discussed extensively. The elements of the gene orders usually represent genes, but the identification of the genes is a cumbersome task. In Pevzner and Tesler (2003) a

method was presented that circumvents this by utilising blocks of unannotated sequence with sufficient sequence similarity as elements of the gene orders. The presented analysis of the rearrangements suggested that breakpoints mainly occur in fragile regions in the genomes giving rise to the fragile breakage model of rearrangement. This is in contrast to the random breakage model of genome rearrangement (Nadeau and Taylor, 1984) assuming every position in a genome having the same probability to break, i.e. to be involved in a rearrangement. The question about which of the models is more appropriate initiated an ongoing discussion (Peng et al., 2006; Sankoff and Trinh, 2005). Various approaches and arguments have been utilised to resolve this issue (Alekseyev and Pevzner, 2007; Bergeron et al., 2008c; Sankoff, 2006), see also Sankoff and Nadeau (2003).

The centromere of a chromosome is a special region of a chromosome of importance for cell division. A variant of sorting by translocations has been studied where only rearrangements are allowed that preserve the centromere of a chromosome in Ozery-Flato and Shamir (2008), motivated by the finding that the loss of the centromere of a chromosome is likely to be lethal.

2.5 Rearrangement median problems

The rearrangement median problem asks for a gene order minimising the sum of some distance measure to a set of given gene orders. The aim of solving this problem is to identify putative ancestral gene orders. It is the most basic case of the maximum parsimony problem for gene order data, i.e. the problem of finding a tree topology and ancestral gene orders minimising the sum of a rearrangement distance measure between the gene orders separated by an edge in the tree (see Section 2.6). In the median problem the tree topology is implicitly given as a star and only one ancestral gene order is to be determined. Solving the median problem for three gene orders is probably the most important and also the most considered case because it is the foundation of many algorithms reconstructing phylogenetic trees from gene orders (see also Section 2.6).

As usual the gene orders are given as signed permutations. Formally, given a set of k signed permutations $\Pi = \{\pi_1, \ldots, \pi_k\}$ of the same length the *median problem* is to find a permutation μ such that $\sum_{i=1}^{k} d(\mu, \pi_i)$ is minimal where d is some rearrangement distance measure, e.g. one of the distance measures reviewed in Section 2.3. The solution of a median problem, i.e. permutation μ, is called *median*. In the literature the breakpoint distance, inversion distance, and recently also the DCJ distance have gained attention. The corresponding median problems are named according to the distance measure, e.g. inversion median problem.

In the following sections the literature on median problems is presented. The median problem for the inversion distance is considered in Section 2.5.1 in detail. Other median problems are treated in Section 2.5.4.

2.5.1 The inversion median problem

The *inversion median problem* (IMP) is to find a signed permutation μ minimising the *score* $S(\Pi, \mu) = \sum_{i=1}^{k} d(\mu, \pi_i)$ for a set of signed permutations $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$ of the same length. The minimum value is denoted by $S(\Pi)$. The IMP is important because it can be used for the inference of phylogenetic trees based on gene order data when only inversions are considered. Solving the IMP is the foundation of several algorithms for the reconstruction of phylogenies based on gene orders, e.g. GRAPPA (Moret et al., 2002a,b, 2001), MGR (Bourque and Pevzner, 2002), and AMGRp (Bernt et al., 2007b) which are described in Section 2.6.

The IMP is an NP-hard problem (Caprara, 2003). Several strategies have been applied in order to solve the problem efficiently, e.g. heuristics (Bernt et al., 2006a; Bourque and Pevzner, 2002), elaborated local search techniques (Lenne et al., 2008), and parallel computing (Bernt et al., 2005; Siepel and Moret, 2001). Also the application of biologically inspired constraints showed promising results. One example is the rEvoluzer heuristic that tries to preserve conserved intervals (Bernt et al., 2006a). A new approach for considering biologically inspired constraint for solving the IMP is presented in Chapter 3.

The methods for solving the IMP can be subdivided into methods searching for a median by iteratively applying inversions and methods using the formal tools for solving the inversion distance problem in order to find a median directly. The former method is the most common, e.g. the exact median solver of Siepel and Moret (2001), the heuristics MGR (Bourque and Pevzner, 2002) and rEvoluzer (Bernt et al., 2005, 2006a). An overview of the available methods is given in Section 2.5.2. The only representative of the second type of methods is Caprara's exact median solver (Caprara, 2003). This median solver is the basis of the algorithms CIP and ECIP described in Section 3.2 and it is an important component of the algorithm TCIP described in Section 3.3. Therefore, it is described in necessary detail in Section 2.5.3.

Before the algorithms for solving the IMP are described, some properties of the IMP are elucidated. The inversion distance is a metric and therefore the triangle inequality holds. That is for three signed permutations π_1, π_2, π_3, $d(\pi_1, \pi_3) \leq d(\pi_1, \pi_2) + d(\pi_2, \pi_3)$ holds. Based on this inequality bounds for the IMP can be derived for the case of three input permutations (Hannenhalli et al., 1995; Siepel and Moret, 2001). A lower bound for an IMP for three permutations π_1, π_2, π_3 is given by:

$$S(\Pi) \geq \left\lceil \frac{d(\pi_1, \pi_2) + d(\pi_2, \pi_3) + d(\pi_1, \pi_3)}{2} \right\rceil . \tag{2.1}$$

A trivial upper bound is:

$$S(\Pi) \leq \min\{(d(\pi_1, \pi_2) + d(\pi_1, \pi_3)), (d(\pi_2, \pi_3) + d(\pi_2, \pi_1)), (d(\pi_3, \pi_1) + d(\pi_3, \pi_2))\}. \quad (2.2)$$

This upper bound is achieved be taking one of the input permutations with the lowest score as median.

The computation of lower bounds for the general case, with $k > 3$, is more intricate. The sum of the distances between consecutive gene orders in a circular ordering of the given gene orders divided by two is a lower bound. Because this holds for every possible circular ordering, a circular ordering maximising the distance sum has to be found in order to derive a good bound. A dynamic programming approach for this problem is described in Bachrach et al. (2005).

2.5.2 Stepwise approaches to the inversion median problem

Most algorithms for the IMP search for a solution by iteratively applying inversions to the permutations. The algorithms differ in the set of applied inversions, i.e. some apply inversions exhaustively and others choose certain subsets of inversions.

One of the first approaches to the IMP is presented in Hannenhalli et al. (1995). The approach is based on the enumeration of the so called d-neighbourhood of a permutation π, which is defined as the set of permutations that can be generated from π by d inversions. Let d_i, with $i \in [1:3]$, denote the inversion distance from π_i to a median. The algorithm enumerates distance triples (d_1, d_2, d_3) which sum up to the lower bound and are compliant with the triangle inequality. The d_i-neighbourhood of permutation π_i is determined for each triple (d_1, d_2, d_3). If the d_i-neighbourhoods have a non empty intersection, a median is found. The approach computes the exact solution if there is one with a score equal to the lower bound. Otherwise, no solution is found.

The first heuristic for the IMP was presented in Sankoff et al. (1996). For each pair of permutations π_i, π_j, with $i, j \in [1:3]$ and $i \neq j$, a sorting inversion scenario is computed. For each permutation σ is this scenario an inversion scenario to the third permutation is computed. Each permutation in these scenario is a candidate solution. With a local search technique the solutions are potentially improved. That is single inversions are applied to the permutations and if this improves the score, the candidate solution is updated. This is repeated until no further improvement is possible.

These remarkable pioneering works are the first approaches to the IMP. But both approaches can not be applied to distant or larger gene orders. Because of their exhaustive enumeration of the d-neighbourhood, respectively the computation of the sorting scenarios with potentially time consuming branch and bound algorithms. Note, both approaches have been extended to incorporate also transpositions.

Siepel's exact median solver (Siepel and Moret, 2001) is a branch and bound algorithm applying recursively inversions starting at an input permutation that fulfils the upper bound given in Equation (2.2). The algorithm uses instead of all possible inversions only those which increase or maintain the distance to the origin permutation. These inversions are enumerated using the method described in Siepel (2002). The permutations resulting from the applied inversions are stored in a priority queue where the bound of a permutation is used for prioritisation. This bound is computed by the distance from the permutation to the origin permutation plus the lower bound (Equation (2.1)) for the median problem consisting of the current permutation and the other two input permutations. Pruning of the search tree is done with the best found solution. The algorithm stops if a solution with a score equal to the global lower bound is found or if all permutations in the priority queue have a bound greater than or equal to the best found solution. The algorithm has an exponential worst case run time of $\mathcal{O}(n^{3d})$, with $d = \min\{d(\pi_1, \pi_2), d(\pi_2, \pi_3), d(\pi_1, \pi_3)\}$.

A variant of the algorithm of Sankoff et al. (1996) with polynomial run time was presented in Earnest-DeYoung et al. (2004). There are two differences. Firstly, only one single permutation σ, which is "halfway along the sorting sequence", is considered. Secondly, only one third of the scenario from σ to the third permutation of the triple is computed. The permutation found at the end of the partial scenario is returned as heuristically determined median. With this simple heuristic the authors were able to consider gene orders containing duplicate genes using an exemplar approach.

The median solvers MGR (Bourque and Pevzner, 2002), rEvoluzer (Bernt et al., 2006a), rEvoluzerII (Bernt et al., 2005), and MedRbyLS (Interian, 2006) are heuristics. In MGR and the rEvoluzer algorithms inversions are applied iteratively starting from all three input permutations until the permutations are transformed into an identical permutation. Instead of the exhaustive enumeration of inversions used in Siepel's median solver, only one promising inversion is selected greedily. Inversions reducing the distance to both of the other two input permutations are used in MGR. If MGR can not find such an inversion, inversions maximising the distance reduction to the other input permutations are chosen. While in MGR the distances to the input permutations are regarded, the current permutations are regarded for distance calculation in rEvoluzer and rEvoluzerII. Furthermore, not only the distance reduction to the other two permutations is regarded but also the distance incrementation to the origin. For further improvement of the optimisation behaviour, MGR uses a look ahead and rEvoluzer a backtracking strategy. The main improvement, besides the parallelisation, of rEvoluzerII compared to rEvoluzer is that several promising inversion path are explored in parallel. This is used to generate a set of equally good solutions. In order to select the inversions efficiently, MGR is restricted to inversions which do not destroy conserved adjacencies. An efficient algorithm for generating nearly all promising inversions (sorting or distance preserving) is used in rEvoluzer and rEvoluzerII. This algorithm is based on the

idea to preserve conserved intervals and properties of the breakpoint graph. This selection procedure has run time $\mathcal{O}(n^2)$. The integration of this inversion selection procedure into Siepel's median solver and MGR resulted in a speedup without affecting the solution quality. In the algorithm presented in (Interian, 2006), inversions are generated from one starting permutation and random inversions are chosen and applied if the score is improved or with probability p.

Another method to improve IMP solvers which iteratively apply inversions has been presented in Arndt and Tang (2007). The idea is to exploit commuting inversions, i.e. sets of inversions where the order of the application does not matter. A brute force enumeration or sampling of the 2^x permutations that can be generated with x commuting inversions has been proposed. Another strategy just applies all commuting inversions at once. The techniques have been applied to Siepel's median solver. Run time improvements for simulated permutations generated with a large number random inversions have been reported. Furthermore, the need to compute all or multiple solutions for the IMP has been pointed out.

In Tang et al. (2004); Tang and Moret (2003b) methods for the handling of gene orders with duplicated or deleted genes with Siepel's median solver are presented. The idea of these approaches is to transform the given gene orders into permutations, i.e. such that each gene occurs exactly once, before solving the median problem. The gene content of the ancestor is determined with a majority argument, i.e. if in the majority of the gene orders a gene is present (respectively absent), it is assumed to be present (respectively absent) in the ancestor. The copy number of duplicated genes is determined in a similar fashion. An exemplar approach is applied in order to equalise the gene content of the three input permutations. Furthermore, all possibilities to delete and insert genes in the input permutations, such that the desired gene content of the ancestor is established, are evaluated. For all possibilities the pairwise distances of the three permutations are computed. The median problem is only solved for those triples with a minimum sum of pairwise distances. In addition to this, improved bounds for the median problem have been introduced.

2.5.3 Caprara's inversion median problem solver

Caprara (2003) presented a branch and bound algorithm solving the IMP exactly. In contrast to other median solvers a median is constructed directly based the formal tools used for the sorting by inversion problem instead of computed by the iterative application of inversions. Caprara's inversion median problem solver is the basis of the algorithms CIP and ECIP presented in Section 3.2 and is used for solving subproblems in algorithm TCIP Section 3.3. Therefore, this algorithm is presented here in detail. Note, besides the branch and bound algorithm presented in the following, an ILP formulation and a heuristic based on the LP relaxation of the ILP formulation have been presented in Caprara (2003). An implementa-

tion of Caprara's algorithm can be found in the GRAPPA package (Moret et al., 2002a,b, 2001).

The central data structure for the branch and bound algorithm is the *multiple breakpoint graph* (MB graph). The MB graph generalises the breakpoint graph to more than two permutations. The MB graph is defined formally in the following. Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of k signed permutations of size n. As usual, let $\pi(i)$, with $i \in [1:n]$, denote the i-th element of a permutation π of length n. Additionally, let $\pi(0) = 0$ and $\pi(n+1) = n+1$. The corresponding MB graph $\mathcal{B}(\Pi)$ is a graph on the node set $V = \{0, 1, \ldots, 2n+1\}$ which possibly has parallel edges with common endpoints. The perfect matching $H = \{(2i-1, 2i) : i = [1:n]\} \cup \{(0, 2n+1)\}$ associated with the vertex set V is called *base matching*. The edge set is defined using the one to one correspondence between signed permutations and perfect matchings M of V, such that $M \cup H$ forms a Hamiltonian cycle. Such perfect matchings are called *permutation matching*s. For a signed permutation π the corresponding permutation matching $M(\pi)$ is defined by

$$M(\pi) = \Big\{ \Big(2|\pi(i)| - \nu\big(\pi(i)\big),\ 2|\pi(i+1)| - 1 + \nu\big(\pi(i+1)\big) \Big) : i \in [0:n] \Big\}, \tag{2.3}$$

where $\nu(\pi(i)) = 0$ if $\pi(i) \geq 0$ and $\nu(\pi(i)) = 1$ if $\pi(i) < 0$. The edge set of the MB graph is given by $\bigcup_{i=1}^{k} M(\pi_i)$.

Two permutation matchings uniquely define a set of *cycles* alternating between the edges of the two matchings. Let $cyc(M(\pi), M(\sigma))$ be the number of *cycles* in the MB graph defined by two permutation matchings corresponding to two permutations π and σ. The heart of Caprara's algorithm is to solve the *cycle median problem* (CMP) which is to find a permutation matching $M(\mu)$ (corresponding to a permutation μ) such that the score of the CMP

$$k \cdot (n+1) - \sum_{i=1}^{k} cyc(M(\mu), M(\pi_i))$$

is minimised. That is a permutation matching is requested maximising the number of cycles. This is inspired by the inequality $d(\pi, \sigma) \geq n+1-cyc(M(\pi), M(\sigma))$ that relates the inversion distance between two permutations π and σ to the number of cycles defined by $M(\pi) \cup M(\sigma)$ (see, e.g. Hannenhalli and Pevzner (1995b)). From this inequality follows that

$$S(\Pi, \mu) = \sum_{i=1}^{k} d(\mu, \pi_i) \geq k(n+1) - \sum cyc(M(\mu), M(\pi_i)).$$

Hence, a solution for the CMP is a lower bound for the IMP. The idea of Caprara's algorithm is to construct solutions for the CMP as candidate solutions for the IMP. This is motivated by the fact that for two permutations π and σ, drawn uniformly and independently from the

set of all permutations of length n, $d(\pi, \sigma) = n + 1 - cyc(\pi, \sigma)$ with a high probability, i.e. the probability is at least $1 - \frac{1}{2n^2} - o(\frac{1}{n^2})$.

The branching operation used in the branch and bound algorithm solving the CMP is the so called edge *contraction operation*. This operation removes an edge together with its adjacent vertices from the MB graph and modifies the affected permutation matchings in an appropriate way. Let M be a perfect matching of a node set V and an edge $e = (i, j) \in \{(k, l) : k, l \in V, k \neq l\}$. Then let

$$M/e = \begin{cases} M \setminus \{(i,j)\} & \text{if } (i,j) \in M \\ M \setminus \{(i,a),(j,b)\} \cup \{(a,b)\} & \text{otherwise, with } (i,a),(j,b) \in M \end{cases}$$

The contraction of an edge e in an MB graph $\mathcal{B}(\Pi)$ yields $\mathcal{B}(\Pi)/e$ with node set $V \setminus \{i, j\}$ and edge set $\bigcup_{i=1}^{k}(M(\pi_i)/e)$. Note, the graph $\mathcal{B}(\Pi)/e$ is not necessarily an MB graph as some $(M(\pi_i)/e) \cup H$ may define more than one cycle. Therefore, the following generalisation of the CMP is considered. Let G be a graph with node set V of size $|V| = 2(n + 1)$ for $n > 0$, H be a perfect matching on V called base matching, and let $E = \{(i, j) : i, j \in V, i \neq j\} \setminus H$. Given k perfect matchings on V, $M_1, \ldots, M_k \subset E$, find a perfect matching $T \subset E$ such that $T \cup H$ defines a Hamiltonian cycle and $k(n + 1) - \sum_{i=1}^{k} cyc(T, M_i)$ is minimised.

Contracting an edge in the MB graph adds the edge to the solution permutation matching which is constructed in a descent towards the leaves of the branch and bound search tree. The algorithm enumerates in a depth first manner all combinations of edges which are permutation matchings and performs a lower bound test after each edge contraction. More precisely, all permutation matchings are enumerated by recursively contracting the edges (i, j) if the last contracted edge is (g, h) and $(h, i) \in H$, for all j that have no incident edge in the partial permutation matching constructed so far. The recursion starts with $i = 0$, i.e. all edges $(0, 1), \ldots, (0, 2n)$ are enumerated. The bound is computed as follows. For a generalised CMP instance with associated perfect matchings $M_1, \ldots, M_k$ an upper bound on the number of cycles for a given instance of the CMP is given by

$$\frac{k \cdot (n+1)}{2} + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{cyc(M_i, M_j)}{k-1}. \tag{2.4}$$

The upper bound on the number of cycles after the contraction of an edge e is given by $|\{i : M_i \ni e\}|$ plus the upper bound given by Equation (2.4) when edge is contracted. A lower bound for the score of the CMP (and also for the IMP) is given by $k(n + 1)$ minus the upper bound on the number of cycles. The lower bound test necessary after each edge contraction, can be achieved in time $\mathcal{O}(nk^2)$.

If a complete permutation matching $M(\sigma)$ is constructed, the CMP score $k \cdot (n+1) - \sum_{i=1}^{k} cyc(M(\sigma), M_i)$ is computed. The current best known solution is updated if the score is smaller than the best solution found so far. The algorithm stops if the branch and bound search tree is exhausted or if the lower bound of the original problem is met.

The branch and bound algorithm finds optimal solutions for the IMP with a small modification. Instead of computing the score with respect to the CMP objective function, when a complete permutation matching $M(\sigma)$ is constructed, the score is computed with respect to the IMP objective function for the permutation σ. That is $\sum_{i=1}^{k} d(\sigma, \pi_i)$ using the inversion distance d.

Example 2.5.1. *In the following a run of Caprara's branch and bound algorithm is exemplified for the three signed permutations* $\pi_1 = (1\ 2\ 3\ 4)$, $\pi_2 = (1\ \text{-}2\ \text{-}4\ 3)$, *and* $\pi_3 = (4\ \text{-}1\ 2\ \text{-}3)$. *The run for these permutations is visualised in Figure 2.6.*

First, the MB graph is computed for the three input permutations (shown in the centre of Figure 2.6 in the area shaded with the darkest grey). The base matching is shown as dashed lines, the permutation matchings of π_1, π_2, *and* π_3 *are shown as blue (respectively red and green) lines. The number of cycles for the pairs of permutation matching is given as small matrix inside the MB graph. For example,* $M(\pi_1)$ *and* $M(\pi_2)$ *forms two cycles, i.e. the cycle containing vertices* 0 *and* 1 *and the cycle which includes the remaining nodes. The other two pairs of permutation matchings forms each one cycle. The initial upper bound on the number of cycles is computed according to Equation* (2.4) *as follows. The left part of the sum is equal to* 7.5 *because* $k = 3$ *and* $n = 4$. *The right part of the sum gives* 2 *because the sum of the number of cycles for all pairs of permutation matchings is* 4. *Thus, for the given problem it is not possible to find a permutation matching forming more than* 9.5 *(i.e.* 9*) cycles with the given permutation matchings. Hence, a lower bound for the CMP (and IMP) is given by* $3 \cdot (4+1) - 9.5 = 5.5$.

Now the first descent to a leaf of the search tree is detailed (shown as bold arrows in the figure). The first edge contraction applied to the MB graph affects the edge $(0, 1)$. *This removes the vertices* 0 *and* 1 *from the graph and the edge set is modified as follows:*

- *the edges connecting the removed vertices* 0 *and* 1 *are removed from the graph, here this affects two edges belonging to* M_1 *and* M_2,
- *the edges, where one endpoint is a removed vertex are modified, here the edges* $(7, 0)$ *and* $(1, 3)$, *are replaced by a new edge* $(7, 3)$.

After the edge contraction operations the upper bound on the number of cycles for the CMP defined by the new graph is according to Equation (2.4) *equal to* 7.5 *(now* $n = 3$ *and one cycle was removed due to the edge contraction). By adding the term* $|\{i : M_i \ni (0, 1)\}| = 2$ *(the MB graph contained the contracted edge twice) this yields an upper bound of* 9.5 *for the original*

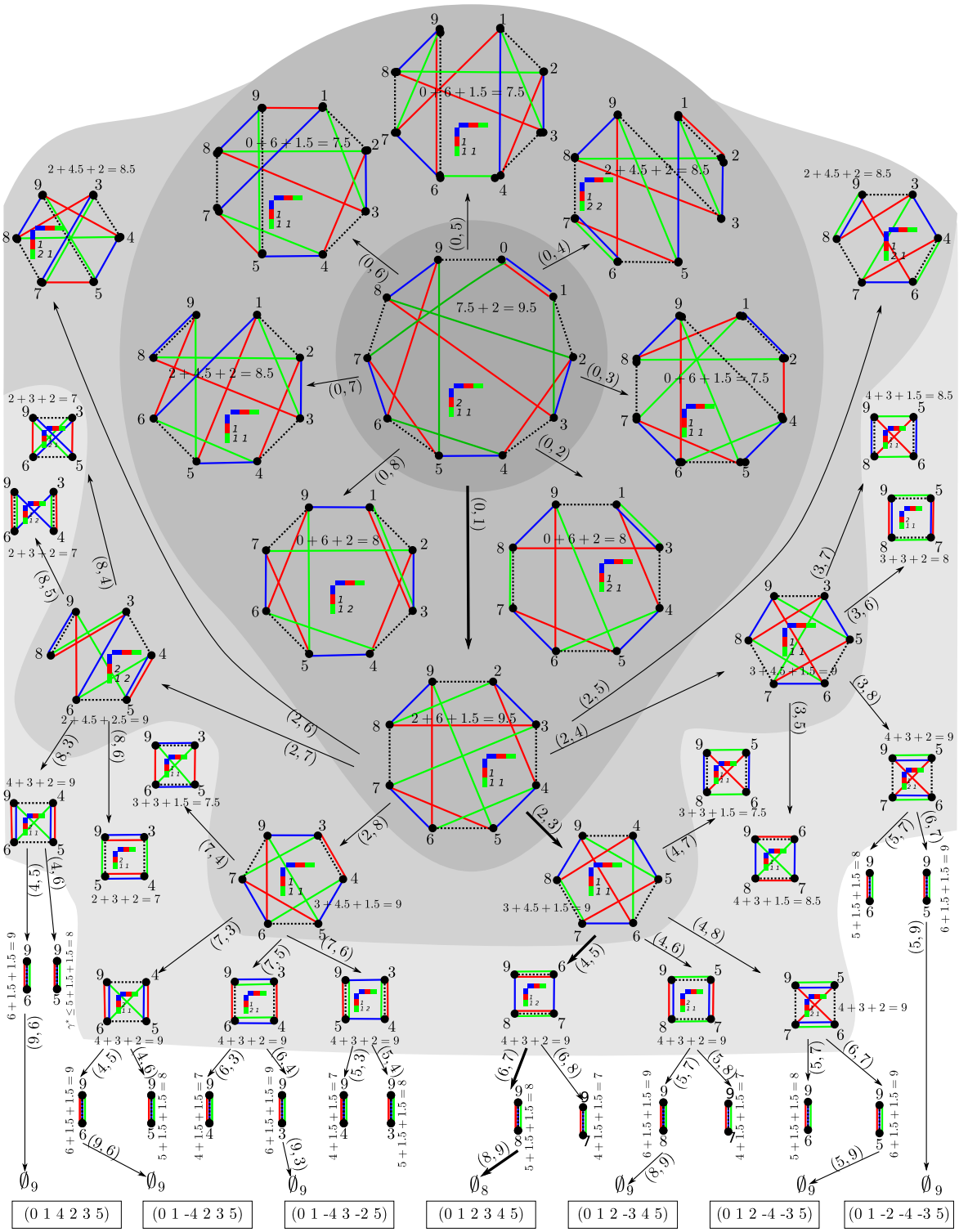


Figure 2.6 – Caprara's inversion median solver for Example 2.5.1; MB graph edges: H dashed, M_1 blue, M_2 red, M_3 green, for each MB graph the matrix of the number of cycles and the computation of the upper bound for the number of cycles is given; arrows indicate edge contractions annotated with the corresponding edge; shaded areas indicate recursion levels; constructed permutations are shown in boxes at the bottom

problem. In the following, the edges $(2,3)$*,* $(4,5)$*,* $(6,7)$*, and finally* $(8,9)$ *are contracted one after the other resulting in an empty graph. The constructed permutation matching, consisting of the contracted edges, forms eight cycles with the input permutation matchings. This is the first constructed solution, i.e. the currently best known. Because the IMP is to be solved, also the score for the IMP is computed. The permutation corresponding to the permutation matching is* (1 2 3 4) *which has a score of seven which is the saved as currently best solution for the IMP. Note, successive edge contraction operations may be imagined as adding adjacencies in a partial permutation. Here, these are the adjacencies* $(1,2)$*,* $(2,3)$*, and* $(3,4)$ *plus the auxiliary adjacencies* $(0,1)$ *and* $(4,5)$ *indicating the first and last element of the permutation.*

The algorithm continues and returns to the graph before the edges $(6,7)$ *and* $(8,9)$ *have been contracted. The next, and only possible, edge contraction affects the edge* $(6,8)$ *leading to a graph with an upper bound for the number of cycles of seven. As this is less than the best known solution, the algorithm does not explore the remaining search tree below this node. Thus, it returns to the state before edge* $(4,5)$ *was contracted. The following contraction of edges* $(4,6)$*,* $(5,7)$*, and* $(8,9)$ *leads to a permutation matching with nine cycles. This is better than the best known solution and equal to the upper bound for the initial MB graph. Thus, a solution with a maximum number of cycles, i.e. minimum CMP score, is found. But because the IMP is to be solved, the update of the best known solution is necessary with respect to the sum of the inversion distances. The corresponding permutation* (1 2 -3 4) *has a score of six. Thus, the permutation is the best known solution seen so far during the run of Caprara's median solver and the old solution has to be replaced. The lower bound of the IMP* $\lceil \frac{3+4+4}{2} \rceil = 6$ *is also met by the found solution. Thus, the recursion can be aborted.*

In the case that all solutions for the IMP have to be computed, the algorithm has to continue in the way sketched above. With the minimal score now at hand many parts of the search tree are skipped. For example all edge contractions including vertex 0 *lead to a graph with an upper bound lower than the minimal score. During the complete run, on the search for all minimal IMP solutions, five more complete permutation matchings with a minimal score for the CMP are enumerated. Three of those also have the minimal score of six with respect to the IMP. These are the permutations* (1 -4 2 3)*,* (1 -2 -4 -3)*, and* (1 2 -4 -3)*.*

The modifications necessary to compute all inversion medians that have been described in the example have been implemented in Caprara's median solver (Bernt et al., 2007b).

A branch and bound algorithm may suffer from the problem that good solutions are found after considerable computation time. In the example shown in Figure 2.6 the branching order leads accidentally to a good solution in the very beginning. With this solution large parts of the search space could be pruned. This is not generally the case. Therefore, the algorithm uses a target value t for the quality of a solution. This target value is initialised with the lower bound of the CMP. The algorithm then searches for a CMP solution with value t. If

such a solution is found, it is minimal and the algorithm stops. Otherwise, no solution with this value exists and the algorithm is started again with t increased by one. The algorithm stops if a solution with a score of the target value is found.

2.5.4 Other median problems

The definition of the rearrangement median can easily be adapted to other rearrangement operation by exchanging of the distance measure.

The breakpoint median problem was one of the earliest considered median problems. This was encouraged to some degree by the absence of other efficient methods for computing rearrangement distances in the past and the breakpoint distance can be computed trivially. Furthermore, is was argued that the independence of the breakpoint distance from rearrangement models is an advantage (see Section 2.2.1). In Bryant (1998); Pe'er and Shamir (1998) it was shown that the median problem for breakpoints is NP-hard for linear as well as circular chromosomes. This holds already for three chromosomes. In Sankoff and Blanchette (1997) solving this problem by a reduction to the travelling salesman problem was proposed. Surprisingly the breakpoint median problem is polynomial time solvable when multichromosomal genomes consisting of linear and circular chromosomes are considered (Tannier et al., 2009).

The inversion median problem constrained to inversion which are symmetric around the origin and terminus of replication was studied in Ohlebusch et al. (2007) where it was shown that this variant can be solved in linear time.

Similar to the comparison of pairs of gene orders, the DCJ rearrangement has gained attention for the median problem. In Adam and Sankoff (2008) the MGR approach was adapted to the DCJ distance. A branch and bound algorithm in the spirit of Caprara's inversion median solver was presented in Xu (2008); Xu and Sankoff (2008). Methods have been presented for decomposing the multiple breakpoint graph into subgraphs which can be solved easier.

Also for the TDRL operation the median problem has been considered (Chaudhuri et al., 2006). Recall, depending on the value of α in the weighting function, α^l where l is the length of the duplicated interval, the TDRL distance is symmetric or asymmetric. For the symmetric case, i.e. $\alpha \geq 2$, the problem was studied before in the field of social choice (see Ailon et al. (2008) and references therein). The NP-hardness is established for $k > 3$. For the median problem for the TDRL distance in the asymmetric case, i.e. $\alpha = 1$, the sum of the distances from the median to the given permutations has to be minimised. That is the direction is from the unknown ancestor to the contemporary gene orders. For this median problem it is also meaningful to consider the case of two input permutations. The hardness of the problem is open. A polynomial time approximation solution for the TDRL median

problem was derived from a reduction to a feedback arc set problem (Chaudhuri et al., 2006).

2.6 Phylogenies from gene arrangements

Several approaches have been proposed for the reconstruction of phylogenetic trees from gene order data. The two main approaches are reviewed in the following with a focus on maximum parsimony methods (see Section 2.6.1). But also the promising character based approaches are briefly introduced in Section 2.6.2.

Distance matrix methods for genome rearrangement analysis are described for example in Wang et al. (2006). The reader is referred to Larget et al. (2005) for Bayesian methods.

2.6.1 Maximum parsimony approaches for gene orders

The *maximum parsimony* or *multiple genome rearrangement* problem for gene orders is defined as follows. Given a set of gene orders Π, the problem is to find a tree $T = (V, E)$ with the given permutations assigned to the leaves and an assignment of permutations to the inner nodes of the tree, such that

$$\sum_{(\pi,\sigma)\in E} d(\pi, \sigma)$$

is minimal. This problem can be considered also for various genome rearrangement distance measures d.

One approach to this problem is to decompose it into the small and the large parsimony problem. For the small parsimony problem a phylogenetic tree T is given and only the assignment of gene orders to the inner nodes is to be determined. The big parsimony problem is also to find the tree, e.g. by an exhaustive enumeration of all trees or, more useful, with some tree search or branch and bound algorithm. This is the theme of breakpoint analysis and GRAPPA described in the following.

The breakpoint analysis approach (Sankoff and Blanchette, 1997; Sankoff et al., 1996) tries to find a minimum phylogenetic tree for the breakpoint distance. The small parsimony problem is solved by repeatedly computing median problems. The gene order of an inner node is determined by solving the median problem for the gene orders of the three adjacent nodes. The method continues to solve median problems as long as the overall score for the given tree is improved. Because only the leaf nodes have assigned gene orders in the beginning a method for initialising the gene orders of the inner nodes are necessary. Different strategies to find an initial assignment of gene orders to the inner nodes of the tree have been proposed. Where assigning an arbitrary permutation to the inner nodes, taking the permutation from a nearest leaf node, or assigning a solution of the median problem

defined by nearest nodes with assigned permutations are the most basic but yet successful strategies. Also more complicated initialisation strategies are available which are specialised for the breakpoint distance. The big parsimony problem is solved by a bounded search over all possible binary trees.

This approach was later reimplemented and adapted to the inversion distance in the GRAPPA software (Moret et al., 2002a,b, 2001). By using sophisticated bounding techniques a million fold speedup compared to the breakpoint analysis implementation was reported. Furthermore, it was shown that the use of the inversion distance is advantageous for phylogeny reconstruction compared to the breakpoint distance. Despite the reasonable efforts to improve the run time, GRAPPA can not be applied to more than about dozen gene orders. The reason for this is the immense number of possible binary trees. One approach for this problem is the disk covering method (Huson et al., 1999) which has been successfully applied to GRAPPA (Tang and Moret, 2003a). This method decomposes the data set and combines the resulting trees found for the smaller portions of the data set. Note, breakpoint analysis and GRAPPA do not solve the multiple genome rearrangement problem exactly because building a tree from exact solutions of the median problem does not ensure a minimum tree.

In MGR (Bourque and Pevzner, 2002) and the amGRP algorithm (Bernt et al., 2007b) the sequential addition approach was adopted for the gene order parsimony problem. That is the given permutations are integrated one after the other into a partially constructed tree. A new permutation is inserted greedily by removing one edge of the partial tree and adding three new edges computed by solving the median problem for the new permutation and the two nodes incident to the removed edge. In a postprocessing step of MGR inversions are applied on the given gene orders which reduce the distance to all other given gene orders until two gene orders converge. If two gene orders have converged, one is removed and the algorithm continues with one gene order less. When no more inversion, which reduces the distance to all other gene orders, can be found, the sequential addition method is applied where for solving the median problems the MGR median solver is used. In contrast to the MGR approach, an exact median solver is used consequently from the very beginning to solve the median problems in amGRP. But a more important difference and main improvement of amGRP compared to all other approaches is to use the set of all medians instead of only one. Therefor the sequential addition approach was extended to use all medians in a bounded tree search. For the handling of large median sets several method for choosing a subset have been analysed.

Recently, Bader et al. (2008) proposed a new method which does not use a median solver for finding an initial tree. Only parsimonious or nearly parsimonious sorting sequences are computed instead. New nodes are integrated into the tree by either connecting it to a node of the tree via a sorting scenario or by removing one edge and connecting the three nodes

via a permutation on a (nearly) sorting scenario between the two nodes. In a second step the initial tree is improved by iteratively solving median problems as usual. It is attempted additionally to improve the topology of the tree by the removal of an edge and the search for a better possibility to reconnect the generated parts of the tree.

The presented algorithms can easily be adapted to definitions of the multiple genome rearrangement problem using different distance measures. Only the availability of a median solver (respectively an algorithm for generating sorting sequences) is necessary for this (Bader et al., 2008).

2.6.2 Gene cluster based phylogeny reconstruction

The so called model-free approaches neither make assumptions on the types of rearrangement operations that occurred during evolution nor on their frequencies. Binary characters are extracted directly from properties of the given gene arrangements instead (hence the property is assumed to be phylogenetically meaningful). In the simplest case such a binary character is defined by the absence or presence of an adjacency of two genes in the given arrangements (Cosner et al., 2000b) or the successor-predecessor relationship of the genes (Bhutkar et al., 2007; Gallut and Barriel, 2002; Gallut et al., 2000; Ma et al., 2006; Tang and Wang, 2005). This definition can easily be extended to sets of genes that either occur consecutively in an arrangement or not (Adam et al., 2007; Stoye and Wittler, 2009). On the other end of the scale a relaxed definition of consecutiveness can be considered (Chauve and Tannier, 2008).

This binary character information can be used in standard phylogenetic methods. A popular approach is to use Fitch's method (Fitch, 1971) to reconstruct the presence vs. absence information at the inner nodes of a given phylogenetic tree (Adam et al., 2007; Bergeron et al., 2004a; Ma et al., 2006; Tang and Wang, 2005). Alternatively, a probability based method is used to reconstruct the ancestral binary states (Adam et al., 2007; Ma et al., 2006). The reconstructed binary information can be used to deduce putative ancestral arrangements. This is complicated due to the fact that possibly no linear order of the genes can fulfil the consecutiveness constraints given by the reconstructed binary characters. Usually greedy heuristics are used to find a linear order of the genes (Adam et al., 2007; Ma et al., 2006), but also computational expensive exact algorithms are available (Chauve and Tannier, 2008; Stoye and Wittler, 2009). The Fitch method can be used to reconstruct parsimonious phylogenetic trees by an iteration over all binary trees.

In Zhao and Bourque (2007) an interesting connection between the model-free approaches and the rearrangement based methods was made. The method searches for patterns in the gene adjacencies of two sets of gene orders corresponding to inversions or transpositions. Rearrangement events are assigned to the edges of a given phylogenetic tree by searching re-

arrangement patterns in the bipartition of the leaves defined by the inner edges. The method can be interpreted as assigning rearrangement events to the edges of a phylogenetic tree by computing intersections of rearrangement scenarios. That is, a rearrangement is assigned to an edge if the corresponding pattern is found for all pairs of gene orders in the corresponding bipartition. This is similar to the `TreeREx` method presented in Section 4.3. But `TreeREx` computes intersections locally, i.e. only for small subtrees. Furthermore, `TreeREx` considers a larger set of rearrangement operations in order to meet the needs of mitochondrial gene order data sets.

3 The preserving inversion median problem

The aim of solving an inversion median problem is to find a putative ancestral gene order explaining the differences in a set of gene arrangements with a scenario which is parsimonious in the number of inversions. Adding further biologically motivated constraints can help to find phylogenetically more meaningful results and it may also lead to more efficient algorithms.

It is known that certain gene groups are preserved during evolution (Lathe et al., 2000). Since it is difficult to determine functionally what a gene group is, it has been proposed to consider common combinatorial structures between gene orders, e.g. conserved intervals or — as used in the following — common intervals to determine groups of genes which have been preserved during evolution, i.e. which appear consecutively in the considered gene orders (see Section 2.2). If a gene group has been conserved during evolution, e.g. when functional constraints have inhibited the destruction of the gene group, the gene group should also be preserved when reconstructing the gene order evolution. The additional constraint that conserved gene groups should be preserved, has been considered for the reconstruction of pairwise rearrangement scenarios (Section 2.4.2). Unfortunately, even computing the preserving inversions distance, i.e. the minimum number of inversions that preserve all common intervals of two given gene orders, is an NP-hard problem (Figeac and Varré, 2004). Without the restriction that common intervals should be preserved, the inversion distance problem, as well as computing a corresponding minimum length sequence of inversions, is polynomial time solvable (Bergeron et al., 2004b; Hannenhalli and Pevzner, 1995b; Tannier et al., 2007) (Section 2.3). Note, the unconstrained inversion median problem is already NP-hard for three given gene orders (Caprara, 2003). Siepel's (Siepel and Moret, 2001) and Caprara's (Caprara, 2003) median solver are the currently available exact algorithms for the inversion median problem (for details the reader is referred to Section 2.5.1).

In this section the inversion median problem is studied under the additional constraint that no common interval is destroyed. This is the problem of finding potential ancestral gene orders for the gene orders of given taxa, such that the corresponding rearrangement scenario has a minimal number of inversions, and where each of the inversions has to preserve the

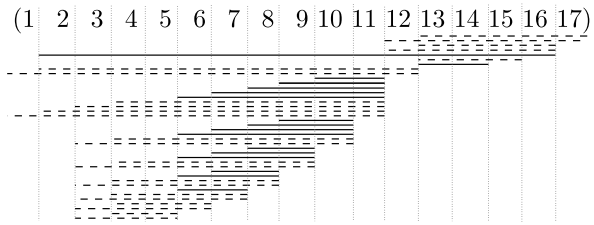


Figure 3.1 – The common intervals of the three mitochondrial gene orders of Example 3.0.1; each common interval is depicted as a line which underlines the contained elements (drawn relative to π_L which is shown on the top); dashed lines indicate the common intervals destroyed by the exact IMP median given in the example

common intervals of the given input gene orders, is studied. The problem of finding such an ancestral gene order is called the *preserving inversion median problem* (pIMP).

Example 3.0.1. *Consider the following three mitochondrial gene orders of Locust (π_L), Silkworm (π_S), and Centipede (π_C), where identical strips of genes are replaced by a single element, taken from* Bergeron and Stoye *(2006):*

- $\pi_L = (1\ 2\ 3\ 5\ 4\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17)$,
- $\pi_S = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 14\ 13\ 15\ 16\ 17)$, *and*
- $\pi_C = (1\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ -2\ 12\ 16\ 13\ 14\ 15\ 17)$.

The three permutations have 40 *common intervals shown in Figure 3.1. For example, the set* $\{13, \ldots, 17\}$ *appears consecutively in all three permutations, this is indicated in Figure 3.1 by a line under the elements* 13 *to* 17. *A median of score* 11 *is given by* $\mu = (1\ 2\ 3\ 4\ -14\ -13\ -12\ -11\ -10\ -9\ -8\ -7\ -6\ -5\ 15\ 16\ 17)$. *The median μ destroys the majority of the common intervals of the three permutations, i.e. 23 common intervals, also shown in Figure 3.1. Besides μ the IMP given by the three permutations has* 13 *medians where the locust gene order π_L is one of them. The locust gene order π_L does not destroy any common interval whereas all other medians destroy at least one common interval. Furthermore, π_L can be reached with* 11 *preserving inversions from the input permutations. Thus, one can argue that π_L should be preferred as median. Note, the set of preserving inversion medians is not generally included in the set of inversion medians.*

In the following two different approaches for the pIMP are presented. The first approach, implemented in the algorithms CIP (Common Interval Preserving) and ECIP (Extended Common Interval Preserving), is to modify Caprara's median solver for the IMP such that the pIMP is solved exactly (see Section 3.2). CIP makes slight changes in the branch and bound algorithm of Caprara's median solver with the effect that median solutions, which destroy common intervals or do not minimise the sum of the preserving inversion distances, are not accepted. The preserving inversion distance measure is computed with the algorithm

described in Bérard et al. (2007). In order to increase the efficiency of the modified algorithm a second version called Extended-CIP (ECIP) is introduced. ECIP avoids computing gene orders that destroy common intervals.

The second approach is inspired by the work presented in Bérard et al. (2007). According to Bérard et al. (2007), parsimonious preserving scenarios, transforming one given gene order into another given gene order, can be computed in polynomial time for certain interesting classes of problem instances. These results are based on the strong interval tree, which is a tree data structure for representing the common intervals of a gene order set. The strong interval tree can be computed in linear time for a constant number of input gene orders. In Section 3.3 strong interval trees are used for the computation of preserving inversion median scenarios, i.e. for solving the preserving inversion median problem (pIMP). The exact algorithm, called TCIP (Tree Common Interval Preserving) for solving the pIMP for given gene orders, is presented and it is shown that TCIP solves certain interesting classes of pIMP instances in linear time.

IMP and pIMP are analysed for mitochondrial gene orders of various taxa of *Metazoa*, *Campanulaceae* chloroplast gene orders, and simulated gene orders in Section 3.4. The importance of the pIMP is empirically established by analysing the number of common intervals and how many common intervals are destroyed by solutions of the IMP for the different data sets. Furthermore, the run time of the three presented pIMP solvers is investigated empirically. For the considered simulated and biological data sets properties of the strong interval trees are analysed with respect to the run time of TCIP. Inversion medians and preserving inversion medians are compared.

3.1 Definitions

A *permutation* $\pi = (\pi(1)\ \pi(2)\ \ldots\ \pi(n))$ of size n is a permutation of the elements in $\{1, 2, \ldots, n\}$. A *signed* permutation of size n is a permutation of size n where every element has an additional "+" or "−" sign that defines its orientation. Sign "+" is usually omitted. In the following, a signed permutation is just called permutation. An *inversion* $\rho(i, j)$, $1 \leq i \leq j \leq n$, applied to a signed permutation π of size n, transforms it into $\pi \circ \rho(i, j) = (\pi(1)\ \ldots\ \pi(i-1)\ -\pi(j)\ \ldots\ -\pi(i)\ \pi(j+1)\ \ldots\ \pi(n))$. When the context is clear, an inversion is identified with the set of elements that are affected. A *sorting inversion scenario* for two signed permutations π and σ is a sequence of inversions $\rho_1, \ldots, \rho_d$ that transforms π into σ. A shortest of such sequences is called *parsimonious* and its length is the *inversion distance* $d(\pi, \sigma)$ between π and σ.

An *interval* of a permutation is a set of consecutive (unsigned) elements of this permutation. Let Π be a set of signed permutations of size n. A *common interval* (Heber and Stoye, 2001b; Uno and Yagiura, 2000) of Π is a subset of $\{1, 2, \ldots, n\}$ being an interval in each

$\pi \in \Pi$. The singletons $\{i\}$, $i \in \{1, 2, \ldots, n\}$, and the set $\{1, 2, \ldots, n\}$ of all elements are called *trivial*. Observe that the signs of the elements are ignored in the definition of common intervals. Let $C(\Pi)$ be the set of all common intervals of Π. Two intervals c and c' *overlap* if $c \cap c' \neq \emptyset$, $c \not\subset c'$, and $c' \not\subset c$. If two intervals do not overlap, they *commute*. A common interval is called a *strong* common interval if it does not overlap with any other common interval. The *strong interval tree* $T(\Pi)$ of Π (Bérard et al., 2007; Eres et al., 2003) is a tree where the nodes are exactly the strong common intervals of Π, such that the root node is the interval containing all elements, the leaves are the singletons, and the edges are defined by the minimal inclusion relation of the intervals, i.e. there is an edge between node c and c' iff $c' \subset c$ and there is no node c'' with $c' \subset c'' \subset c$. As the common intervals of a set of permutations Π are independent of the signs of the elements, and therefore also the strong common intervals, the topology of $T(\Pi)$ does not depend on the signs of the elements as well. The strong interval tree can be computed in time $\mathcal{O}(kn)$ for k signed permutations of size n (Bergeron et al., 2008a).

An inversion ρ applied to one of the permutations $\pi \in \Pi$ *preserves* the common intervals of Π if it does not destroy any common interval $c \in C(\Pi)$, i.e. $C(\Pi) = C(\Pi \cup \{\pi \circ \rho\})$. If there exists a common interval $c \in C(\Pi)$, that does not exist after applying the inversion, i.e. $c \notin C(\Pi \cup \{\pi \circ \rho\})$, the inversion does *not preserve* $C(\Pi)$. An inversion, which preserves the common intervals of Π when applied to $\pi_i \in \Pi$, is called *preserving with respect to* Π and π_i. The inversion is simply called *preserving with respect* to Π when π_i is obvious and just *preserving* if Π and π_i are obvious. The *preserving inversion distance* $d_C(\pi, \sigma)$ between two signed permutations π and σ is the minimum number of preserving (with respect to $\{\pi, \sigma\}$) inversions necessary to transform π into σ (in Bérard et al. (2007) this distance is denoted as perfect inversion distance).

The *inversion median problem* (IMP) deals with finding for a set of signed permutations $\Pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$ a signed permutation μ which has a minimal total inversion distance to the permutations in Π, i.e. the so called *score* $\sum_{i=1}^{k} d(\mu, \pi_i)$ has to be minimal. In this case permutation μ is called *median* of Π. For a signed permutation σ, a *scenario for* Π *to* σ defines for each permutation in Π a sequence of inversions to σ. The inversions applied to transform $\pi_i \in \Pi$ into σ are referred to as π_i*-scenario* or *i-th scenario*. The sequence of permutations traversed from permutation $\pi_i \in \Pi$ towards σ is called a *trace* of the scenario, or more specific: the π_i*-trace* of the scenario. If σ is a median, i.e. the total number of inversions in the scenario is minimal, the scenario is called *median scenario*. The *preserving inversion median problem* (pIMP) is defined as the IMP with the additional restriction that only inversions, which are preserving with respect to the k given permutations, are allowed. The corresponding permutation μ is called *preserving median* or *median of the pIMP*. Observe that $C(\Pi) = C(\Pi \cup \{\mu\})$ holds for medians μ of the pIMP. A slightly different definition of the pIMP was used in Bernt et al. (2006b), where for the computation of the preserving

inversion distance only pairwise common intervals of an input permutation and the median were taken into account. A scenario for a set of permutations Π to a permutation σ is *preserving* if each of its inversions is preserving with respect to Π. If σ is a preserving median, this scenario is also called *preserving median scenario*.

3.2 CIP - ECIP

The algorithms CIP and ECIP which solve the preserving inversion median problem exactly are introduced in this section. Both algorithms are modifications of Caprara's median solver (for details see Section 2.5.3). The simple modification of the algorithm CIP is presented in the next section.

3.2.1 CIP

Caprara's median solver enumerates permutation matchings in its branch and bound search, i.e. permutations which are candidate solutions for the IMP. Each time a complete permutation matching is constructed, Caprara's algorithm checks if the IMP score of the corresponding permutation is smaller than the best score found so far. Algorithm CIP modifies this check in order to find preserving medians. The first modification is that only candidate solutions σ for updating the best known solution are accepted which preserve the common intervals of Π, i.e. where $C(\Pi) = C(\Pi \cup \{\sigma\})$. Clearly, this check is required before updating the best known solution. Additionally, the inversion distance measure $\sum_{i=1}^{k} d(\sigma, \pi_i)$ is replaced by the corresponding structure preserving inversion distance measure $\sum_{i=1}^{k} d_C(\sigma, \pi_i)$. This is necessary because $C(\pi) = C(\Pi \cup \{\gamma\})$ must hold for every permutation γ in a π_i-trace to σ.

3.2.2 ECIP

A potentially more efficient method to adapt Caprara's median solver is to constrain the branch and bound search. The branching operation of the algorithm, i.e. the addition of an edge to the partial permutation matching, may be regarded as adding a new element to the partial permutation by fixing an adjacency. Recall, there is a one-to-one relationship between edges of a permutation matching and adjacencies of the corresponding permutation. Thus, the algorithm can be adapted to produce only common interval preserving permutations by forbidding certain branching steps, i.e. by forbidding adjacencies of the permutation which are impossible when preserving common intervals. Again, the inversion distance measure has to be replaced by the corresponding common interval preserving inversion distance measure for updating the best known solution. In the remainder of this section the constraints that can be used for common intervals are described. Static and dynamic constraints are distinguished.

Static constraints

The static constraints are impossible adjacencies which are independent of the decisions made during the assembly of a candidate median gene order. Let n denote the length of the considered permutations. The static constraints can be described by an $n \times n$ zero-one matrix $\mathcal{M}$ which can be computed before the run of the algorithm. Let $e, f \in [1:n]$ be two elements of the permutations in Π. Then $\mathcal{M} = (m_{ef})$ is defined as follows.

$$m_{ef} = \begin{cases} 1 & \text{if the adjacency of } e \text{ and } f \text{ is allowed} \\ 0 & \text{else} \end{cases}$$

Recall, the elements' orientation is not regarded for the definition of common intervals. Hence, the orientation of elements e and f is not regarded in the equation given above. In the following it is described how the static constraints, stored in the matrix $\mathcal{M}$, are derived. It can be observed that a single common interval does not imply static constraints because every element can be moved freely within this interval, and thus any adjacency with an element from inside or outside is possible. But, if there are overlapping common intervals, their elements can not be moved freely within both intervals. This is formally shown in Theorem 3.2.1. A few additional definitions are necessary at first.

In the following it is assumed that the identity permutation $\iota \in \Pi$ without loss of generality. By this assumption the elements of each common interval of Π are successive elements. That is given a common interval c, there exist l, r, with $1 \leq l \leq r \leq n$, such that $[l:r] = c$. The *overlap graph* of a set of common intervals C is the graph $G(C) = (C, E)$ with node set C and an edge between nodes c and c' if they overlap. To describe the static constraints that can be derived from overlapping intervals, consider a connected component of $G(C)$ and let $C' \subseteq C$ be the nodes of this component. Clearly, the set of elements appearing in at least one common interval in C' is a subinterval $[r:s]$ of $[1:n]$. Ergo there exist indices $r = i_1, \ldots, i_k \leq s$, with $k > 1$ and $1 \leq i_1 < \ldots < i_k < i_{k+1} = s + 1$ such that for each subinterval $I_j = [i_j : i_{j+1} - 1]$, $j \in [1:k]$ and each common interval c in C' either $I_j \subset c$ or $I_j \cap c = \emptyset$ and the subintervals are maximal with this property. The partition into the subintervals is unique. Obviously, two elements $i \in I_h$ and $j \in I_l$, $1 \leq h \leq l \leq k$ are allowed to be adjacent only when $|h-l| \leq 1$. Moreover, an element $i \in I_1 \cup I_k$ is allowed to be adjacent to elements in $[1 : i_1 - 1] \cup [i_{k+1} : n]$. Clearly, all these constraints are static constraints. It is not hard to show that the subintervals defined above commute with all common intervals, i.e. they are strong.

Theorem 3.2.1. *Let Π be a set of signed permutations of length n and C' be the nodes of a connected component of the overlap graph $G(C(\Pi))$. Furthermore, let $(I_1, \ldots, I_k)$ be*

the partition of the elements of the component into maximal intervals, such that $I_j \subset c$ *or* $I_j \cap c = \emptyset$ *holds for all* $c \in C'$ *and* $j \in [1:k]$.

A permutation σ *is preserving for* C' *iff for all adjacencies* (i, j) *of* σ

i) with $i \in I_l$ *and* $j \in I_h$ $|l - h| \leq 1$ *holds and*

ii) if one element is not in $\bigcup_{i=1}^{k} I_i$ *and the other is then the included element is in* I_1 *or* I_k.

Proof. Let σ be a permutation, I_0 denote the elements of σ to the left of $\bigcup_{i=1}^{k} I_i$ and I_{k+1} denote the elements to the right.

Let σ be a permutation that is preserving for the common intervals in a connected component C' of $G(C(\Pi))$. It is shown that adjacent elements in σ are from the same or from adjacent intervals of the partition $(I_0, \ldots, I_{k+1})$. For now consider only adjacencies of elements which are both included in the component.

Let $h \in [0 : k-1]$ and $l = h + 2$, i.e. $|h - l| > 1$, and let e_0, e_1, e_2 be elements of a permutation with $e_i \in I_{h+i}$. By definition $J = I_h \cup I_{h+1}$ and $K = I_{h+1} \cup I_{h+2}$ are also two common intervals. It holds that $e_0, e_1 \in J$, $e_2 \notin J$ and $e_1, e_2 \in K$, $e_0 \notin K$. Now assume that the elements e_0 and e_2 are adjacent in a permutation π that is preserving with respect to the common intervals of Π. The following configurations of permutations with this adjacency are possible:

i) $(\ldots\ e_1\ \ldots\ e_0\ e_2\ \ldots)$ or $(\ldots\ e_2\ e_0\ \ldots\ e_1\ \ldots)$: then the interval containing e_1 and e_2 also contains e_0 contradicting that K is a common interval,

ii) $(\ldots\ e_0\ e_2\ \ldots\ e_1\ \ldots)$ or $(\ldots\ e_1\ \ldots\ e_2\ e_0\ \ldots)$: then the interval containing e_0 and e_1 also contains e_2 contradicting that J is a common interval.

Thus, elements from intervals I_h and I_{h+2} can not be adjacent in a preserving permutation.

Now the general case of $|h - l| > 2$ is considered. Without loss of generality $h < l$ is assumed. By definition it holds that $\bigcup_{i \in [h+1:l-1]} I_i$ is a common interval. Thus, the general case can be shown as above when the intervals I_h, $\bigcup_{i \in [h+1:l-1]} I_i$, and I_l are considered.

To complete the proof it is shown that if all adjacencies (e, f) of a permutation σ are from the same or adjacent subintervals, σ is preserving for C'. Let e and f be two elements which are adjacent in σ and c be a common interval from C'. The following cases are possible:

i) If $e, f \in I_l$, with $l \in [0 : k+1]$, no common interval of C' is destroyed because $\forall c \in C'$: either $c \cap I_l = \emptyset$ or $c \supset I_l$.

ii) If $e \in I_l$ and $f \in I_{l+1}$, with $l \in [0 : k]$, in none of the possible cases $c \supset I_l \wedge c \supset I_{l+1}$, $c \supset I_l \wedge c \cap I_{l+1} = \emptyset$, $c \supset I_{l+1} \wedge c \cap I_l = \emptyset$, and $c \cap I_l = \emptyset \wedge c \cap I_{l+1} = \emptyset$ the common interval is destroyed.

iii) The case that $e \in I_{l+1}$ and $f \in I_l$ is analogous.

If one of the cases applies for all adjacencies, σ is preserving for C'. □

Example 3.2.1. *Let* C *be the set of three intervals* c_1, c_2, c_3 *common to a set of permutations of size eight (the actual set of permutations is irrelevant for the example), with* $c_1 = [1 : 4]$,

$c_2 = [3 : 6]$, and $c_3 = [5 : 8]$. $G(C)$ consists of a single connected component containing all three intervals. Then the intervals $I_1 = [1 : 2]$, $I_2 = [3 : 4]$, $I_3 = [5 : 6]$, and $I_4 = [7 : 8]$ form the maximal partition with the desired property $\forall i \in [1 : 4], j \in [1 : 3] : I_i \subset c_j \vee I_i \cap c_j = \emptyset$. Then

$$\mathcal{M} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

That is 24 of the 56 possible adjacencies are forbidden by static constraints in this example. Now consider a run of Caprara's median solver where the partial permutation (1 3 5 7 8) is constructed. All adjacencies are allowed by the static constraints given by $\mathcal{M}$, i.e. $\mathcal{M}_{1,3} = 1$, $\mathcal{M}_{3,5} = 1$, $\mathcal{M}_{5,7} = 1$, and $\mathcal{M}_{7,8} = 1$ but appending any of the remaining elements 2 or 4 is forbidden by the static constraints. Appending 2 would result in a permutations where c_1 is no interval because $5, 7, 8$ are in between. Appending element 4 would destroy c_2.

Dynamic constraints

In Example 3.2.1 further branching was forbidden very late in the recursion after almost the complete permutation was constructed. This can be improved by the use of a different type of constraints. They are given by the observation that choosing an element from an interval implies that all elements of this interval have to be chosen before elements that are not included in this interval. This is a direct consequence of the definition of common intervals. Recall that the branch and bound algorithm constructs a candidate permutation by successively adding adjacencies during the branching steps. Thus, these constraints are dynamic because they depend on the decisions made during a run of the algorithm. In ECIP the dynamic constraints are implemented in the following way. If the element chosen in a branching step is from one of the subintervals of the form I_h, $h \in [1 : k]$ (as described above), it is clear that the next elements must be chosen from the same interval until no element remains in I_h. The dynamic constraints are sufficient only in combination with the static constraints described in the last section because the adjacency of elements from non adjacent subintervals of a connected component is not forbidden by the dynamic constraints. All these dynamic constraints are computed during the run of the modified version of Caprara's median solver used for ECIP.

Example 3.2.2. *Consider the order of appending elements to the partial permutation given in Example 3.2.1. After the first element, i.e.* 1*, the dynamic constraints forbid that any element but* 2 *is appended in the next recursion step. If element* 2 *is appended in the next step, all of the remaining elements are allowed by the dynamic constraints, but the static constraints allow only* 3 *or* 4*. In this way the permutations* $\pi = (\pi(1), \ldots, \pi(8))$ *with* $\{\pi(2i-1), \pi(2i)\} = I_i$*, for* $i \in [1:4]$*, and the permutations with* $\{\pi(2i-1), \pi(2i)\} = I_{5-i}$*, for* $i \in [1:4]$*, are enumerated successively. All these permutations are preserving. Which of the preserving permutations is minimal has to be determined by calculating the preserving inversion distance to the input permutations.*

Algorithms

The sketch of Algorithm ECIP and the necessary modifications in Caprara's branch and bound IMP median solver are presented in Algorithms 1 to 3.

Algorithm 1: ECIP: Preprocessing

Input: permutations π_1, π_2, π_3 of size n
Output: $\mathcal{M}$: matrix describing the static constraints
data structures l and R for the computation of the dynamic constraints

```
1  initialise M: ∀e, f ∈ {1,...,n} : m_ef ← 1;
2  C ← common intervals of {π1, π2, π3};
3  find the connected components in the overlap graph G(C);
4  forall connected components c do
5      determine the i_j (1 ≤ j ≤ k + 1);
6      I_j ← [i_j : i_{j+1} − 1]  ∀j ∈ [1 : k];
7      forall I_h and I_l with |h − l| > 1 do
8          ∀e ∈ I_h, ∀f ∈ I_l : m_ef ← 0;
9      forall j ∈ [1 : k] do
10         ∀e ∈ I_j : R_c[e] ← j;
11     forall j ∈ [1 : k] do
12         l_c[j] ← i_{j+1} − i_j;
```

Algorithm 1 describes the preprocessing phase of ECIP in which the matrix $\mathcal{M}$ for checking the static constraints and two data structures, needed for checking the dynamic constraints, are initialised. This algorithm is run once before the start of the modified branch and bound algorithm. First, the values of $\mathcal{M}$ are initialised to 1, i.e. everything is allowed, and the common intervals and the connected components of the common interval overlap graph are computed (lines 2 and 3). Then for each component the k subintervals I_j, for $j \in [1:k]$ are initialised. This can be done efficiently by a simple line sweep if the common intervals are given sorted by their start and end index. The static constraints implied by the currently

considered connected component are updated by forbidding adjacencies between elements of $e \in I_l$ and $f \in I_h$ for all $|h - l| > 1$, with $h, l \in [1 : k]$ (line 8). Furthermore, data structures R_c and l_c are initialised for each connected component c of the overlap graph (lines 10 and 12). R_c stores the index of the subinterval I_j which it is part of for each element of the permutations. The data structure l_c stores the number of elements of each I_j which are still available in the branching step. The $l_c[j]$, with $j \in [1 : k]$, are initialised with the size of the intervals I_j.

Algorithm 2: ECIP: Modification of branching in Caprara's IMP solver

```
  // f is to be appended to the partial permutation ending with e
1 if branching allowed for (e, f) then
2     forall connected components c with f ∈ c do
3         l_c[R_c[f]] ← l_c[R_c[f]] − 1;
4     branch;
5     forall connected components c with f ∈ c do
6         l_c[R_c[f]] ← l_c[R_c[f]] + 1;
```

Algorithm 2 shows the necessary modifications for the branching step in Caprara's median solver. The code of the original branching procedure in Caprara's median solver, i.e. appending and removing elements to or from the partial permutation and the recursion, is indicated in line 4. The modification consists of wrapping this part of Caprara's median solver into a test, such that branching is only allowed if appending the next element to the current partial permutation can lead to a preserving permutation, i.e. the static and dynamic constraints are satisfied. This check is realised by Algorithm 3 (executed in line 1). If branching is not allowed, Caprara's median solver will try to append another element to the partial permutation or return to the last recursion level when all possibilities to append an element are exhausted. Otherwise, if branching is allowed, the data structure l_c has to be adjusted before and after the branching for each connected component c of the overlap graph. If element j is appended, the number of elements of the subinterval which contains j, i.e. $R_c[j]$, which are not included in the partial permutation, has to be reduced by one before the branching (line 2) and increased by one after the branching (line 5).

The algorithm deciding whether a branching step, i.e. an adjacency (e, f), is allowed is shown in Algorithm 3. First, the static constraints are checked (see line 2). If $m_{ef} = 0$, the algorithm returns $false$, i.e. the branching is not allowed. This is executed before checking the dynamic constraints because the access to $\mathcal{M}$ is in $\mathcal{O}(1)$. If no static constraint rejects the adjacency (e, f), the dynamic constraints have to be checked. This happens separately for each component (see lines 3 to 5).

Algorithm 3: ECIP: Branching restriction in Caprara's median solver

Input: (e, f): a potential adjacency
Data structures $\mathcal{M}, R, l$ (see Algorithm 1 and Algorithm 2)
Output: true if branching is allowed, false otherwise

```
   // check static constraints
1  if m_ef = 0 then
2      return false;
   // check dynamic constraints
3  forall connected components c with e ∈ c do
       // check number of elements to be placed
4      if R_c[e] ≠ R_c[f] ∧ l_c[R_c[e]] > 0) then
5          return false;
6  return true;
```

In case elements e and f are from the same subinterval; or included in different subintervals and all elements from the subinterval that contains e are already chosen (see line 6) holds for all components, the branching step, i.e. adding adjacency (e, f), is compliant with the dynamic constraints. Note that the check, if a new subinterval is entered by appending f, is realised by checking for equality of the subinterval indices. This is sufficient because subinterval index differences greater than 1 and the case that element e or f is not in the component are already treated by the static constraints. Furthermore, if the subinterval indices are equal, no dynamic constraints are implied. If the dynamic constraints are violated for a component, false is returned. Otherwise, if the adjacency is compliant with the dynamic constraints of all components (and the static constraints), true is returned.

$\mathcal{M}$ is implemented as a matrix. The data structures l_c and R_c can be realised as arrays in the following way. R_c is implemented as an array of length n storing for the component c the index of the subinterval that contains it and l_c is an array of length k (i.e. number of subintervals). Thus, the access to all used data structures can be assumed to be in $\mathcal{O}(1)$. Therefore, the number of components, respectively subintervals per component is the main factor for the run time of the algorithms.

The components of the overlap graph $G(C(\Pi))$ correspond to chains of irreducible common intervals of Π. It is well known that the number of irreducible intervals is smaller than n (Heber et al., 2009). Obviously, the algorithms iterate once (respectively twice) over the connected components of the overlap graph and within each iteration a constant number of constant time operations are executed. Thus, these algorithms are very efficient. i.e. Algorithm 2 and Algorithm 3 have run time in $\mathcal{O}(n)$. This is important because they are called very often in the modified branch and bound pIMP solver. Algorithm 1 has polynomial run time because an iteration over the connected components is necessary and additionally

iterations over the pairs of elements for pairs of subintervals is performed. The run time is not analysed here in greater detail because the preprocessing time is strongly dominated by the branch and bound algorithm solving the pIMP which has an exponential worst case run time.

CIP and ECIP can also solve the pIMP for circular gene orders. This can be accomplished by rotating and inverting the complete gene orders such that all start with element 1. Solving the pIMP for the obtained linear gene orders gives also a solution for the circular gene orders. The correctness is shown in Section 3.3.5.

It should be noted that it is enough to consider only irreducible common intervals in order to compute the static and dynamic constraints (Bérard et al., 2004, Proposition1). This is advantageous because the number of irreducible common intervals is always smaller than n, whereas the number of common intervals is only smaller than $\binom{n}{2}$ in general (Heber et al., 2009).

3.3 TCIP: solving the preserving inversion median problem

in Bérard et al. (2007) strong interval trees for pairs of permutations have been introduced and based on this data structure algorithms for the sorting by preserving inversions and for computing the preserving inversion distance have been presented (see Section 2.4.2). In this section a generalisation of strong interval trees for more than two permutations is presented (see Section 3.3.1). After giving the theoretical foundations of the method in Section 3.3.3, methods for solving the pIMP based on properties of this tree are given in Section 3.3.4. The methods are used in the proposed algorithm (TCIP) for the pIMP (Section 3.3.5).

3.3.1 k-signed strong interval trees

Let π be a permutation and $\mathcal{I} = (I_1, \ldots, I_m)$ be an (ordered) partition of the elements of π into intervals. Similarly as in Bérard et al. (2007) the (unsigned) *quotient permutation* of a signed permutation π associated with $\mathcal{I}$ — denoted $\pi_{|\mathcal{I}}$ — is defined as an unsigned permutation of $\{1, 2, \ldots, m\}$, such that x precedes y in $\pi_{|\mathcal{I}}$ iff the elements of I_x precede the elements of I_y in π. The latter precedence relation is well defined since each set in $\mathcal{I}$ is an interval of π. Note, the quotient permutation depends on the order of the sets in $\mathcal{I}$ but not on the signs of the elements of π. A quotient permutation is called *increasing linear* if $\pi_{|\mathcal{I}}$ is the identity permutation $(1 \ \ldots \ m)$, *decreasing linear* if $\pi_{|\mathcal{I}}$ is the inverse of the identity $(m \ \ldots \ 1)$, and *prime* otherwise. A quotient permutation is called *linear* if it is increasing linear or decreasing linear.

Example 3.3.1. *Let* $\pi = (3\ 7\ 1\ 4\ 5\ 6\ 2)$ *be a permutation, and let* $\mathcal{I} = (I_1, I_2, I_3)$ *with* $I_1 = \{6, 2\}$, $I_2 = \{3, 7, 1\}$, *and* $I_3 = \{4, 5\}$ *be a partition of* π *into intervals. The quotient permutation* $\pi_{|\mathcal{I}}$ *of* π *is* $(2\ 3\ 1)$. *Since it is not linear, it is prime.*

From the definition of a strong interval tree $T(\Pi)$, the children of an inner node define a partition of this node into strong common intervals. Using the definition of quotient permutations, the property linear or prime is assigned to the inner nodes of a strong interval tree as follows. Recall, each node of a strong interval tree corresponds to an interval in each permutation $\pi_i \in \Pi$. Consider an inner node I of the strong interval tree. The children of node I define a partition $\mathcal{I}$ of this interval into maximal strong intervals. Without loss of generality assume that the sets in the partition $(I_1, \ldots, I_m)$ are ordered, such that I_x precedes I_y in the first permutation π_1 for $x < y$. Consider the corresponding k quotient permutations $\Pi_{|I} = (\pi_{1|I}, \ldots, \pi_{k|I})$ that are induced by this ordering scheme for interval I in the permutations $\Pi = (\pi_1, \ldots, \pi_k)$. Each of these quotient permutations is either prime or linear. If all k quotient permutations are linear, the inner node I is called *linear*. Otherwise, it is called *prime*. Leaf nodes are assumed to be linear. The quotient permutation for the interval I in π_1 is increasing linear by the assumed ordering of the intervals of $\mathcal{I}$ with respect to π_1. But note, a node is linear or prime regardless of the choice of the permutation defining the ordering of the intervals in $\mathcal{I}$.

Example 3.3.2. *Let* $\pi_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$, $\pi_2 = (4\ 5\ \text{-}8\ 3\ \text{-}2\ \text{-}1\ \text{-}7\ \text{-}6)$, *and* $\pi_3 = (\text{-}7\ \text{-}6\ \text{-}5\ \text{-}4\ 1\ 2\ \text{-}3\ 8)$ *be the three input permutations. Their non-trivial strong common intervals are* $\{1, 2, 3\}$, $\{4, 5\}$, *and* $\{6, 7\}$. *The strong interval tree is depicted in Figure 3.2.*

Consider the node I *in the strong interval tree, which corresponds to the strong common interval* $\{1, \ldots, 8\}$. *The children of node* I *are* $I_1 = \{1, 2, 3\}$, $I_2 = \{4, 5\}$, $I_3 = \{6, 7\}$, *and* $I_4 = \{8\}$. *They are ordered relative to* π_1, *i.e.* I_1 *precedes* I_2, I_2 *precedes* I_3, *and* I_3 *precedes* I_4 *in* π_1. $\mathcal{I} = (I_1, I_2, I_3, I_4)$ *is an ordered partition of the node into intervals. The quotient permutations that are induced for* I *are* $\pi_{1|\mathcal{I}} = (1\ 2\ 3\ 4)$, $\pi_{2|\mathcal{I}} = (2\ 4\ 1\ 3)$, *and* $\pi_{3|\mathcal{I}} = (3\ 2\ 1\ 4)$. *Node* I *is prime because* $\pi_{2|\mathcal{I}}$ *and* $\pi_{3|\mathcal{I}}$ *are not linear, i.e. prime.*

Now consider the node J *in the strong interval tree, corresponding to the strong common interval* $\{1, 2, 3\}$. *The children of node* J *are* $J_1 = \{1\}$, $J_2 = \{2\}$, *and* $J_3 = \{3\}$, *which are again ordered relative to* π_1. *Set* $\mathcal{J} = (J_1, J_2, J_3)$ *is a partition of the node into intervals. The quotient permutations that are induced for* J *are* $\pi_{1|\mathcal{J}} = (1\ 2\ 3)$, $\pi_{2|\mathcal{J}} = (3\ 2\ 1)$, *and* $\pi_{3|\mathcal{J}} = (1\ 2\ 3)$. *All three quotient permutations are linear, hence the node* J *is linear.*

In the following a variation of the strong interval tree which requires an order on a given set of permutations is defined. Therefore, it is assumed that a sequence $\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$ of permutations is given. To simplify matters, the notation Π is used to denote the sequence of permutations and also the corresponding set of permutations. It will be clear from the

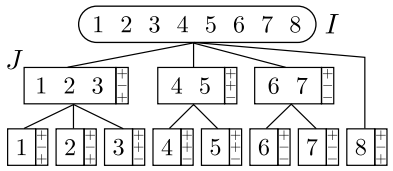


Figure 3.2 – The 3-signed strong interval tree for the permutations of the Examples 3.3.2 and 3.3.3; linear nodes are depicted with rectangular boxes and prime nodes with rounded boxes; signs indicate the 3-sign of a node, with $s(1), s(2), s(3)$ from top to bottom

context whether Π is a sequence or a set, if this is not explicitly stated. Some definitions given for a set of permutations will also be used for a sequence of permutations when the meaning is clear.

A tuple $s = (s(1), \ldots, s(k))$, where $s(i) \in \{+, -\}$, is called a *k-sign*. The *inverted* tuple of s, i.e. the tuple for which each sign is inverted, is denoted by $-s$. The *k-signed strong interval tree* $T^k(\Pi)$ is the strong interval tree of the sequence $\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$ of k permutations that has assigned a k-sign to some of its nodes, such that

i) each leaf has a k-sign $(s(1), \ldots, s(k))$ where $s(i)$ is the sign of the corresponding element of the input permutation π_i,

ii) the h-th entry in the k-sign of a linear node is "+" if the quotient permutation of the node in π_h is increasing and it is "−" if the quotient permutation is decreasing (recall that the sets in the partition are ordered with respect to π_1), and

iii) a prime node inherits the k-sign of its parent if and only if the parent is linear.

The k-sign assigned to a node I is denoted by $s(I)$. Note that no k-sign is assigned to prime nodes which have a prime node parent as well as to a prime node which is the root node of the strong interval tree. A k-signed strong interval tree is called *unambiguous* if every prime node has a linear parent and *ambiguous* otherwise. Examples of 3-signed strong interval trees are given in the following sections.

Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of permutations and I a node in $T(\Pi)$ with child nodes $I_1, \ldots, I_m$. Then $\mathcal{I} = (I_1, \ldots, I_m)$ is a partition of I into intervals. Let $s_1, \ldots, s_m$ be the k-signs of the child nodes $(I_1, \ldots, I_m)$. The *signed quotient permutation* of the i-th permutation π_i associated with $\mathcal{I}, s_1, \ldots, s_m$ — denoted $\pi_{i|\mathcal{I},s_1,\ldots,s_m}$ — is defined as a signed permutation of $\{1, 2, \ldots, m\}$, such that x precedes y in $\pi_{i|\mathcal{I},s_1,\ldots,s_m}$ iff the elements of I_x precede the elements of I_y in π_i and the sign of an element e in the signed quotient permutation equals the i-th sign of s_e, i.e. $s_e(i)$. For the signed quotient permutations of a prime node the unknown signs of prime child nodes are marked with a $\pm$.

Example 3.3.3. *Consider the same input permutations and quotient permutations as in Example 3.3.2. The* 3*-signed strong interval tree is depicted in Figure 3.2. In order to compute the signed quotient permutations of a node, the* 3*-signs of the child nodes must be computed. The three child nodes of node* J *are leaf nodes and therefore the* 3*-signs are determined by the signs of the corresponding elements. As the elements* 1 *and* 2 *(corresponding to child nodes* J_1 *and* J_2*) are positive in* π_1 *and* π_3 *and negative in* π_2*, it holds that* $s_1 = s_2 = (+,-,+)$*. The* 3*-sign of the third child node* J_3 *is* $s_3 = (+,+,-)$ *because the corresponding element* 3 *is positive in* π_1 *and* π_2 *and negative in* π_3*. With this, the signed quotient permutations for node* J *are given by* $\pi_{1|J,s_1,s_2,s_3} =$ (1 2 3)*,* $\pi_{2|J,s_1,s_2,s_3} =$ (3 -2 -1) *and* $\pi_{3|J,s_1,s_2,s_3} =$ (1 2 -3)*. For example in* $\pi_{2|J,s_1,s_2,s_3}$ *element* 3 *is positive because* $s_3(2) = +$ *and elements* 1 *and* 2 *are negative because* $s_1(2) = s_2(2) = -$*.*

Similarly the signed quotient permutations of node I *are computed. The* 3*-sign of the child node* I_4*, which is a leaf node, is computed in the same way as described above and is given by* $s_4 = (+,-,+)$*. The* 3*-signs of the other child nodes are determined by their quotient permutations, i.e. if they are increasing or decreasing linear. This is exemplified for the child node* $I_1 = J$ *in the following. The quotient permutations* $\pi_{1|J}$ *and* $\pi_{3|J}$ *are linear increasing and* $\pi_{2|J}$ *is linear decreasing. Therefore, the* 3*-sign* $s_1 = (+,-,+)$*. The other two* 3*-signs are calculated in the same way and are given by* $s_2 = (+,+,-)$ *and* $s_3 = (+,-,-)$*. The signed quotient permutations of* I *are* $\pi_{1|I,s_1,\ldots,s_4} =$ (1 2 3 4)*,* $\pi_{2|I,s_1,\ldots,s_4} =$ (2 -4 -1 -3)*, and* $\pi_{3|I,s_1,\ldots,s_4} =$ (-3 -2 1 4)*.*

Note also that node I *itself has no* k*-sign because it is prime and has no linear parent node.*

The k-signed strong interval tree data structure can be built in time $\mathcal{O}(kn)$ for a set of k permutations of length n using the algorithms presented in Bergeron et al. (2008a). A k-signed strong interval tree has $\mathcal{O}(n)$ nodes (Bérard et al., 2007), which makes it a suitable data structure for designing efficient algorithms.

Two more formal notations are introduced before presenting the theoretical results in the next section which are the basis of algorithm TCIP. For the comparison of two k-signs the following notations are used. Given two k-signs $s = (s(1), \ldots, s(k))$ and $s' = (s'(1), \ldots, s'(k))$, the set of indices, where the k-signs are unequal, is defined as $\mathcal{U}(s, s') = \{i : s(i) \neq s'(i), i \in [1:k]\}$ and the set of indices where the k-signs are equal is defined as $\mathcal{E}(s, s') = \{i : s(i) = s'(i), i \in [1:k]\}$. When the context is clear, $\mathcal{U}$ and $\mathcal{E}$ is used. Note that $\mathcal{U}$ and $\mathcal{E}$ define a bipartition of $\{1, \ldots, k\}$. That is $\mathcal{U} \cup \mathcal{E} = \{1, \ldots, k\}$ and $\mathcal{U} \cap \mathcal{E} = \emptyset$. Hence, given one of the sets the other is given implicitly, i.e. $\mathcal{U} = \{1, \ldots, k\} \setminus \mathcal{E}$, respectively $\mathcal{E} = \{1, \ldots, k\} \setminus \mathcal{U}$.

Let a k-signed strong interval tree $T(\Pi)$ of a sequence of permutations be given. A sequence of permutations Π' is called *consistent* with $T(\Pi)$ if each node in $T(\Pi)$ is also a common interval of Π'. A sequence of permutations Π' with $C(\Pi) = C(\Pi \cup \Pi')$ is consistent with $T(\Pi)$. If a sequence of permutations Π' is consistent with $T(\Pi)$ and $|\Pi| = |\Pi'| = k$, the

quotient permutations and the k-signs for $T(\Pi)$ can also be computed with respect to the permutations in Π'. Clearly, the quotient permutations and the k-signs can be different depending whether they are computed with respect to Π or Π'. Without loss of generality, the sets in the partition $(I_1, \ldots, I_m)$ are still ordered, such that I_x precedes I_y in permutation $\pi_1 \in \Pi$. This tree is denoted $T^k(\Pi, \Pi')$, but in the following $T^k(\Pi)$ or $T(\Pi)$ is used for briefness when it is clear with respect to which sequence of permutations the k-signs are defined.

3.3.2 The oriented inversion median problem

In the following a slightly modified version of the IMP is introduced which occurs as a subproblem when solving the pIMP with TCIP. Solving this version of the IMP is a key element in the Algorithm TCIP. Given k permutations $(\pi_1, \ldots, \pi_k)$ of length n, the *oriented inversion median problem* (oIMP) is to find a median permutation μ of length n, such that for a given tuple of signs $(s_1, s_2, \ldots, s_k)$ the *score* $\sum_{i=1}^{k} d(\pi_i, s_i \circ \mu)$ is minimal, where $s_i \circ \mu = \mu$ if $s_i = +$ and $s_i \circ \mu = -\mu$ if $s_i = -$ (i.e. the whole signed permutation is inverted). The oIMP is NP-hard, as it contains the IMP for minimising $\sum_{i=1}^{k} d(\pi_i, \mu)$ as a special case when the tuple of signs is $\{+\}^k$. Note that the oIMP for a given sign tuple $(s_1, \ldots, s_k)$ and signed permutations $\pi_1, \ldots, \pi_k$, i.e. minimising $\sum_{i=1}^{k} d(\pi_i, s_i \circ \mu)$, can be solved as a standard median problem where $\sum_{i=1}^{k} d(s_i \circ \pi_i, \mu)$ is minimised. Hence, an oIMP instance can be solved with a standard IMP solver, e.g. Caprara's exact median solver (Caprara, 2003).

3.3.3 Theoretical results

The theoretical results, providing the basis for the design of algorithm TCIP are shown in this section. First, based on results of Bérard et al. (2007), the set of preserving inversions is specified in terms of properties of the k-signed strong interval trees (Proposition 3.3.1 and Theorem 3.3.2). The effects of preserving inversions of k-signed inversions are given in Propositions 3.3.3 and 3.3.4. Based on this Theorem 3.3.5 gives a set of inversions that has to be applied in any preserving inversion median scenario. At the end of this subsection Propositions 3.3.6 and 3.3.7 give additional properties of the IMP which are necessary for the handling of prime nodes in TCIP.

Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of permutations and μ be a permutation that is preserving with respect to Π. Let $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be a sequence of permutations where π'_i is on the i-th trace of a preserving scenario of Π to μ where the i-th trace is defined to be the π_i-trace. Let $\Pi'' = (\pi'_1, \ldots, \pi'_i \circ \rho, \ldots, \pi'_k)$ be the sequence of permutations after an inversion ρ of a preserving scenario has been applied to π'_i. Note that Π' as well as Π'' is consistent with $T^k(\Pi)$ because $C(\Pi) = C(\Pi') = C(\Pi'')$. Due to the application of ρ, the k-signed strong interval tree is changed from $T^k(\Pi, \Pi')$ to $T^k(\Pi, \Pi'')$. As Π' and Π'' are consistent with

$T^k(\Pi)$, the topology of $T^k(\Pi, \Pi')$ and $T^k(\Pi, \Pi'')$ is the same, because the node set is defined by the strong common intervals of $C(\Pi) = C(\Pi') = C(\Pi'')$, only the quotient permutations and k-signs of nodes may be different. The application of a preserving scenario for Π to μ will transform $T^k(\Pi, \Pi)$ successively into $T^k(\Pi, \Pi_\mu)$, where $\Pi_\mu = \{\mu\}^k$. In $T^k(\Pi, \Pi_\mu)$ the k-signed quotient permutations of every node I are equal, either increasing linear or decreasing linear and consequently the k-sign of every node is either $\{+\}^k$ or $\{-\}^k$.

Remember that an inversion can be identified by the set of elements which it reverses and a node of the k-signed strong interval tree by the set of elements which are contained in the corresponding strong common interval. Therefore, nodes, intervals, and inversions can be compared and set operations may be applied. The following proposition shows the relation of preserving inversions and the strong interval tree. It is a modified version of Proposition 2 in Bérard et al. (2007).

Proposition 3.3.1. *Let Π be a set of permutations and σ a permutation of the same length, with $C(\Pi) = C(\Pi \cup \{\sigma\})$. A scenario for Π to σ is preserving with respect to Π iff each of the inversions in the scenario is either a node of the strong interval tree $T(\Pi)$ or the union of children of a prime node of $T(\Pi)$.*

Clearly, a scenario for Π to σ is preserving with respect to $C(\Pi)$ iff the scenarios for all $\pi \in \Pi$ to σ are preserving with respect to $C(\Pi)$. Proposition 2 in Bérard et al. (2007) shows that a scenario S for a permutation π to a permutation σ is preserving with respect to $C(\{\pi, \sigma\})$ if and only if each of the inversions of S is either a vertex of $T(\{\pi, \sigma\})$ or the union of children of a prime vertex of $T(\{\pi, \sigma\})$. As $C(\{\pi, \sigma\}) \subseteq C(\Pi)$ the results of Bérard et al. (2007) can not be applied directly. The following definition and theorem, which are also taken from Bérard et al. (2007), are necessary for proving Proposition 3.3.1.

Definition 3.3.1. *(Reformulation of Definition 9 in Bérard et al. (2007)) Let F be a set of common intervals of a set of permutations Π. The* closure *F^* of F is the smallest set of common intervals of Π that contains F, all trivial common intervals of Π, and for any $I_1 \in F^*$ and $I_2 \in F^*$, if I_1 and I_2 overlap, then $I_1 \cup I_2$, $I_1 \cap I_2$, $I_1 \setminus I_2$, and $I_2 \setminus I_1$ belong to F^*.*

The only difference to Definition 9 in Bérard et al. (2007) is the use of a set of permutations instead of a single permutation (and the identity). This does not affect the definition of the closure.

Theorem 3.3.2. *(Reformulation of Theorem 5 in Bérard et al. (2007)) Let F be a set of common intervals of a set of permutations Π. Let $\pi \in \Pi$ be a permutation and $\mathcal{I} = (I_1, \ldots, I_k)$ be the partition of π into maximal strong intervals of F^* other than π itself. Then, exactly one of the following is true:*

1. *either every union of consecutive elements* $I = \{i, \ldots, j\}$ *of* $\pi_{|\mathcal{I}}$ *is a common interval of* $\pi_{|\mathcal{I}}$ *and* $K = \bigcup_{i \leq h \leq j} I_h$ *belongs to* F^**, or*

2. *no union of intervals of* $\mathcal{I}$ *belongs to* F^**.*

Nodes, which are compliant with property 1), are linear and nodes compliant with 2) are prime. Furthermore, note that a subset of the elements of a permutation is a common interval iff it is a vertex of the strong interval tree or a union of consecutive children of a linear node. This follows from Theorem 3.3.2 and the property that quotient permutations "inherit" the common intervals of the permutations (Bérard et al., 2007). That is if $\mathcal{I} = (I_1, \ldots, I_m)$ is a partition of the elements of π into common intervals, $\{j, \ldots, h\}$ is a common interval of the quotient permutation $\pi_{|\mathcal{I}}$ iff $\bigcup_{j \leq i \leq h} I_i$ is a common interval of π.

In contrast to Theorem 5 in Bérard et al. (2007), F is the set of common intervals of a set of permutations Π and $\pi \in \Pi$ here. In Theorem 5 in Bérard et al. (2007) a permutation π (and the identity) and F is a subset of the common intervals of π and the identity. Without loss of generality it is assumed that the identity is one of the permutations in Π. Therefore it certainly holds that $C(\Pi)$ is a subset of the common intervals of π and the identity. The equality of the theorems follows. As the set of all common intervals $C(\Pi)$ for a set of permutations Π is equal to its closure, Theorem 3.3.2 is in particular true for $F = F^* = C(\Pi)$. Note that the following proof of Proposition 3.3.1 is basically the same as the proof of Proposition 2 in Bérard et al. (2007), where only pairwise preserving scenarios were considered.

Proof of Proposition 3.3.1. Let S be a scenario of Π to σ. Suppose that every inversion of S is either a node of $T(\Pi)$ or the union of children of a prime node of $T(\Pi)$. Let ρ be an inversion of S. If there is a node I in $T(\Pi)$ with $I = \rho$, I is a strong common interval and hence ρ commutes with every common interval of Π. Now, assume that ρ is the union of children of a prime node J. ρ obviously commutes with any common interval not contained in J and with any common interval contained in a child of J. Hence, it remains to show that ρ commutes with any common interval being a union of children of J, but there are none by definition of a prime node. It follows that ρ commutes with every common interval of π and S is a preserving scenario (with respect to Π).

Conversely, suppose that S is a scenario of $\pi \in \Pi$ to σ, that is preserving with respect to Π. Let ρ be an inversion of S, and consider the partition $I_1, \ldots, I_k$ of ρ in which the part containing an element x of I is the largest strong common interval included in ρ and that contains x. If $k \geq 2$, $I_1, \ldots, I_k$ must all be children of the same parent J in $T(\Pi)$; otherwise, I would not commute with the nodes of $T(\Pi)$ that are parents of I_js. If J is a linear node, ρ must be equal to J; otherwise, ρ would overlap an interval formed by a leftover child of J and one of the intervals $I_1, \ldots, I_k$ and such an interval is a common interval of Π by points 1

and 2 of Theorem 3.3.2 using $F^* = C(\Pi)$. Therefore, either $k = 1$ and ρ is a node of $T(\Pi)$, or $k > 1$ and the node J must be prime. □

By Proposition 3.3.1 it is clear which inversions are allowed in a preserving scenario. A preserving inversion changes the signed quotient permutations and k-signs of some nodes and may transform a prime node into a linear node (and vice versa). These effects are a consequence of Proposition 3.3.1 and the definition of the signed quotient permutations of the nodes of a k-signed strong interval tree. Nevertheless, these consequences are given explicitly in the following propositions. First the changes on the signed quotient permutations of the nodes are given formally.

Proposition 3.3.3. *Let Π, $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be sequences of k permutations of the same length such that Π' is consistent with $T(\Pi)$. Let ρ be a preserving inversion applied to π'_i, with $i \in [1:k]$, and $\Pi'' = (\pi''_1, \ldots, \pi''_k)$ denote the resulting sequence of permutations with $\pi'_i \circ \rho = \pi''_i$ and $\pi'_j = \pi''_j$ for all $j \in [1:k]$ with $j \neq i$. Let I be a non leaf node in the strong interval tree $T^k(\Pi, \Pi')$ (respectively $T^k(\Pi, \Pi'')$) with child nodes $I = (I_1, \ldots, I_m)$. The signed quotient permutations for node I are $(\pi'_{1|I}, \ldots, \pi'_{k|I})$ in $T^k(\Pi, \Pi')$ and $(\pi''_{1|I}, \ldots, \pi''_{k|I})$ for node I in $T^k(\Pi, \Pi'')$.*

It holds that

$$\pi''_{i|I} = \begin{cases} \pi'_{i|I} \circ \rho_{|I} \text{ with } \rho_{|I} = \{1, \ldots, m\} & \text{if } \rho \supseteq I \\ \pi'_{i|I} \circ \rho_{|I} \text{ with } \rho_{|I} \subset \{1, \ldots, m\} & \text{if } \rho = \bigcup_{r \in \rho_{|I}} I_r \subset I \\ \pi'_{i|I} & \text{else} \end{cases}$$

and $\pi'_{j|I} = \pi''_{j|I}$ for $j \neq i$.

Proof. By the definition of signed quotient permutations, changes in the i-th permutation only change the i-th signed quotient permutation of a node. Therefore, $\pi'_{j|I} = \pi''_{j|I}$ holds for all $j \neq i$.

The comparison of the i-th quotient permutation of a node I in $T^k(\Pi, \Pi')$ and $T^k(\Pi, \Pi'')$ is as follows.

i) Let $\rho \cap I = \emptyset$. Then the order of the elements of I in π_i and consequently the order of the subintervals of I is not changed in π. Thus, $\pi'_{h|I} = \pi''_{h|I}$ holds.

ii) Now, let $\rho \supseteq I$. Because the order of the elements in ρ is inverted and $I \subseteq \rho$, the order of the elements of I is inverted. That is the order of the subintervals of I is inverted in π. Because the argument applies recursively for the children, the signs of the quotient permutation are inverted also. Hence, $\pi'_{h|I} = -\pi''_{h|I}$, i.e. $\rho_{|I} = \{1, \ldots, m\}$.

iii) $\rho \subset I$ is subdivided into the two cases which are possible for preserving inversions.

a) ρ is a union of children of (a prime node) I. Then the order of the elements of ρ is inverted in π_i and consequently the order of the subintervals in I, which are included in the inversion, is inverted. Because case ii) holds recursively for the affected children also the signs of the corresponding elements in the quotient permutation are inverted. Hence, $\pi'_{h|I} = \pi''_{h|I} \circ \rho_{|I}$ with $\rho_{|I} = \{j : I_j \subseteq \rho, j \in \{1, \ldots, m\}\}$.

b) Finally, in case that $\rho \subset I_k$ for some $k \in \{1, \ldots, m\}$, neither the order of the subintervals of I in π_i nor the quotient permutation of a child of I is changed. Thus, $\pi'_{i|I} = \pi''_{i|I}$ holds ($\rho_{|I} = \emptyset$).

□

That is for each preserving inversion ρ there is a corresponding inversion for one of the signed quotient permutations of a node I in the strong interval tree. Such an inversion is called the *quotient inversion* corresponding to ρ, denoted by $\rho_{|I}$, or just quotient inversion if the context is clear.

The next proposition describes the impact of a preserving inversion on the k-signs of the nodes of a k-signed strong interval tree. This completes the necessary theoretical treatment of the consequences of preserving inversions on the quotient permutations and k-signs of the nodes of a strong interval tree.

Proposition 3.3.4. *Let Π, $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be sequences of k permutations of the same length such that Π' is consistent with $T(\Pi)$. Let ρ be a preserving inversion applied to π'_i, with $i \in [1 : k]$, and $\Pi'' = (\pi''_1, \ldots, \pi''_k)$ denote the resulting sequence of permutations with $\pi'_i \circ \rho = \pi''_i$ and $\pi'_j = \pi''_j$ for all $j \in [1 : k]$ with $j \neq i$. Let I be a linear node in the strong interval tree $T^k(\Pi, \Pi')$ with k-sign s. If $\rho \cap I = \emptyset$ or $\rho \subset I$ holds, I has the same k-sign in $T^k(\Pi, \Pi')$ and $T^k(\Pi, \Pi'')$. If $I \subseteq \rho$, the i-th sign of $s(I)$ in $T^k(\Pi, \Pi')$ is inverse to the i-th sign of $s(I)$ in $T^k(\Pi, \Pi'')$ and the other signs of $s(I)$ are the same in both trees.*

Proof. Let $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be a sequence of permutations consistent with Π. Let $\Pi'' = (\pi'_1, \ldots, \pi'_i \circ \rho, \ldots, \pi'_k)$ be the sequence of permutations after a preserving inversion ρ has been applied to the i-th permutation. If an inversion is applied to the i-th permutation, only the i-th element of the k-sign of a node is affected. The comparison of k-signs in a node I of $T^k(\Pi, \Pi')$ and $T^k(\Pi, \Pi'')$ is as follows.

i) Let $\rho \cap I = \emptyset$. If I is a leaf node, the corresponding element is not influenced and the k-sign is not changed. Otherwise if I is not a leaf node, no quotient permutation of I is changed and therefore no k-sign is changed.

ii) Let $\rho \subset I$. As ρ is a proper subset of I, I cannot be a leaf node. Due to Proposition 3.3.1 the preserving inversion ρ can only include all children of a linear node or any union of children of a prime node. Thus, because I is a linear node, ρ is either a child of I or

is completely contained in a child of I. Then the quotient permutations of I are not changed as ρ does not change the order of the children of I.

iii) Let $I \subseteq \rho$. If I is a leaf node, the i-th entry of the k-sign is inverted, as in π_i' and π_i'' the corresponding element is inverted. If I is not a leaf node, the corresponding quotient permutation of node I is inverted because the order of the child nodes of I is inverted in π_i''. Therefore, the i-th entry of the k-sign is inverted.

□

With Propositions 3.3.3 and 3.3.4 the consequences of preserving inversions for the properties of nodes in the corresponding strong interval tree can be described. This is an important step for identifying the inversions leading to a preserving median scenario in the next sections.

The following theorem specifies which preserving inversions have to be applied in any preserving median scenario.

Theorem 3.3.5. *(The generalised median parity theorem) Let I be a node of the strong interval tree $T^k(\Pi)$ of a sequence of signed permutations $\Pi = (\pi_1, \ldots, \pi_k)$ of the same length, that has a linear parent node J or is the root node of $T^k(\Pi)$. Let $\mathcal{E} = \mathcal{E}(s(I), s(J))$ and $\mathcal{U} = \mathcal{U}(s(I), s(J))$, where $s(I)$ denotes the k-sign of node I and $s(J)$ the k-sign of node J, respectively $\{+\}^k$ if I is the root node of $T^k(\Pi)$. If $|\mathcal{E}| < |\mathcal{U}|$ ($|\mathcal{U}| < |\mathcal{E}|$), the inversions of I in all i-th scenarios, with $i \in \mathcal{E}$ ($i \in \mathcal{U}$ respectively), belong to any preserving median scenario for Π. If $|\mathcal{E}| = |\mathcal{U}|$, the inversions of I in all i-th scenarios with either all $i \in \mathcal{E}$ or all $i \in \mathcal{U}$ belong to any preserving median scenario for Π.*

Proof. Let S be a preserving median scenario for Π and let I be a node in the strong interval tree $T^k(\Pi)$. Let $R \subseteq \{1, \ldots, k\}$ be the set specifying the scenarios in S which include the inversion of I, i.e. $i \in R$ iff the inversion of I is in the i-th scenario of S.

Assume that $R \neq \mathcal{E}$ and $R \neq \mathcal{U}$, i.e. $s(I) \neq s(J)$ and $s(I) \neq -s(J)$ holds after applying all inversions of I in all i-th scenarios with $i \in R$. By Propositions 3.3.1 and 3.3.4, $s(I) \neq s(J)$ and $s(I) \neq -s(J)$ still holds after applying all remaining inversions from S, as an inversion that inverts a sign in $s(J)$ also inverts the corresponding sign in $s(I)$ and all other inversions neither change $s(I)$ nor $s(J)$. Therefore, $s(I) = s(J)$ or $s(I) = -s(J)$ is not achievable by S. But then S can not be a preserving scenario for Π because it is clear that after applying all inversions of a preserving scenario the k-sign of each node must be equal to $\{+\}^k$ or equal to $\{-\}^k$. From this contradiction it follows immediately that $R = \mathcal{U}$ or $R = \mathcal{E}$ must hold, i.e. the inversions of I in all i-th scenarios with either all $i \in \mathcal{E}$ or all $i \in \mathcal{U}$ belong to each preserving scenario for Π.

It remains to show which of the choices can lead to a preserving scenario for Π of minimal length. If $|\mathcal{U}| < |\mathcal{E}|$, only $R = \mathcal{U}$ can lead to a preserving median scenario because $R = \mathcal{E}$

would not be parsimonious. Similarly only $R = \mathcal{E}$ leads to a preserving median scenario in the case that $\mathcal{E}$ is smaller than $\mathcal{U}$. If the sets $\mathcal{E}$ and $\mathcal{U}$ have equal size, both choices — applying inversions in all j-th scenarios for $j \in \mathcal{E}$, $j \in \mathcal{U}$ respectively — are parsimonious.

Consider the case of I being a linear root node of $T^k(\Pi)$. If $\mathcal{U} \leq \mathcal{E}$ ($\mathcal{E} \leq \mathcal{U}$) holds for $s(J) = \{+\}^k$, $\mathcal{E} \leq \mathcal{U}$ ($\mathcal{U} \leq \mathcal{E}$) holds for $s(J) = \{-\}^k$. Because $\mathcal{U}(s(I), \{+\}^k) = \mathcal{E}(s(I), \{-\}^k)$ and $\mathcal{E}(s(I), \{+\}^k) = \mathcal{U}(s(I), \{-\}^k)$ it is enough to consider $\{+\}^k$. □

By the generalised median parity theorem it is clear which inversions have to be applied on linear nodes with linear parent. If a k-signed strong interval tree has no prime nodes, a preserving inversion median scenario is immediately implied by the theorem. This is detailed in Section 3.3.4.

For the treatment of prime nodes with algorithm TCIP the following propositions are necessary. They describe the possible difference of the scores of two inversion median problems that differ by inversions of a single elements or of the complete permutation. Furthermore, the relation of the sets of medians is given for inversion median problems of that kind with maximal score difference. Note, in the following $\mathcal{U}$ and $\mathcal{E}$ are used to identify a set of permutations where an inversion is applied, respectively not applied in the following. Before, $\mathcal{U}$ and $\mathcal{E}$ have been used to identify differences and similarities of k-signs. The reason is implicitly given in Theorem 3.3.5, which shows that sets $\mathcal{U}$ and $\mathcal{E}$, give the differences and similarities of two k-signs and at the same time they identify the set permutations, where an inversion is to be applied.

Proposition 3.3.6. *Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of signed permutations of length n with the set of medians M with score ξ and let ρ be an inversion either of a single element $e \in [1:n]$ or of a complete permutation. Given sets $\mathcal{U} \subseteq \{1, \ldots, k\}$ and $\mathcal{E} = \{1, \ldots, k\} \setminus \mathcal{U}$, let $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be the sequence of signed permutations with $\pi'_i = \pi_i \circ \rho$ if $i \in \mathcal{U}$ and $\pi'_i = \pi_i$ if $i \in \mathcal{E}$. Let M' denote the set of medians of Π' with score ξ'.*

Then $|\xi - \xi'| \leq \min(|\mathcal{U}|, |\mathcal{E}|)$ holds. Furthermore, in case of $\xi - \xi' = \min(|\mathcal{U}|, |\mathcal{E}|)$ it holds

i) $\forall \mu \in M' : \mu \in M$ if $|\mathcal{U}| \leq |\mathcal{E}|$ (i.e. $M' \subseteq M$) and

ii) $\forall \mu \in M' : \rho \circ \mu \in M$ if $|\mathcal{E}| \leq |\mathcal{U}|$ (i.e. $\{\mu \circ \rho : \mu \in M'\} \subseteq M$).

Proof. First of all, note that an inversion of a single element or a complete permutation commutes with all other inversions and can be inserted into any scenario. Consider the case of $\xi \geq \xi'$. Assume that $\xi - \xi' > \min(|\mathcal{U}|, |\mathcal{E}|)$, i.e. $\xi' + \min(|\mathcal{U}|, |\mathcal{E}|) < \xi$.

i) Let $|\mathcal{U}| \leq |\mathcal{E}|$, i.e. $\xi' + |\mathcal{U}| < \xi$. A scenario for Π to μ can be constructed from a median scenario of Π' to $\mu \in M'$ by replacing the traces from π'_i to μ, where $i \in \mathcal{U}$ by $(\pi_i = \pi'_i \circ \rho, \pi'_i, \ldots, \mu)$. Note that the traces from π'_i to μ, with $i \in \mathcal{E}$, already are traces for π_i to μ as $\pi'_i = \pi_i$. This yields a scenario for Π to μ with score $\xi' + |\mathcal{U}|$. Since $\xi' + |\mathcal{U}| < \xi$ this contradicts that ξ is the score of the median problem defined by Π.

ii) Let $|\mathcal{E}| \leq |\mathcal{U}|$, i.e. $\xi' + |\mathcal{E}| < \xi$. A scenario for Π to $\mu \circ \rho$ can be constructed from a median scenario of Π' to μ by replacing the traces from π'_i to μ with $i \in \mathcal{E}$ by $(\pi_i = \pi'_i \circ \rho \circ \rho, \pi'_i \circ \rho, \ldots, \mu \circ \rho)$ and the traces from π'_i to μ, with $i \in \mathcal{U}$ by $(\pi_i = \pi'_i \circ \rho, \ldots, \mu \circ \rho)$. This results in a scenario for Π to $\mu \circ \rho$ with score $\xi' + |\mathcal{E}|$. Since $\xi' + |\mathcal{E}| < \xi$ this contradicts that ξ is the score of the median problem defined by Π. Therefore, $|\xi - \xi'| \leq \min(|\mathcal{U}|, |\mathcal{E}|)$ holds.

The second claim follows immediately from the constructions given above. The case that $\xi' \geq \xi$ can be shown analogously by swapping Π with Π', π_i with π'_i, ξ with ξ', and M with M' in the above. But note that in this case also the median set relation is swapped. That is if $\xi' - \xi = \min(|\mathcal{U}|, |\mathcal{E}|)$ then it holds that if $|\mathcal{U}| \leq |\mathcal{E}|$ then $\forall \mu \in M : \mu \in M'$ and if $|\mathcal{E}| \leq |\mathcal{U}|$ then $\forall \mu \in M : \rho \circ \mu \in M'$.

□

The following proposition generalises this result by allowing more than one inversion to be applied to different subsets of a sequence of permutations.

Proposition 3.3.7. *Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of signed permutations of length n with the set of medians M and score ξ. Let $(\rho_1, \ldots, \rho_l)$ be a sequence of pairwise disjoint inversions where each inversion ρ_j, with $j \in [1 : l]$, either inverts a single element $e \in [1 : n]$ or a complete permutation. Given a sequence of sets $(\mathcal{U}_1, \ldots, \mathcal{U}_l)$ and $(\mathcal{E}_1, \ldots, \mathcal{E}_l)$, with $\mathcal{U}_j \subseteq \{1, \ldots, k\}$ and $\mathcal{E}_j = \{1, \ldots, k\} \setminus \mathcal{U}_j$, let $\Pi' = (\pi'_1, \ldots, \pi'_k)$ be the sequence of permutations where π'_i, with $i \in [1 : k]$, is generated from π_i by applying the set of inversions $\{\rho_j : i \in \mathcal{U}_j, j \in [i : l]\}$. Let M' denote the set of medians of Π' with score ξ'.*

Then $|\xi - \xi'| \leq \sum_{j=1}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ holds. Furthermore, let $R_{|\mathcal{E}|<|\mathcal{U}|} = \{j : |\mathcal{E}_j| \leq |\mathcal{U}_j|, j \in [1 : l]\}$ and $R_{|\mathcal{E}|=|\mathcal{U}|} = \{j : |\mathcal{E}_j| = |\mathcal{U}_j|, j \in [1 : l]\}$. If $\xi - \xi' = \sum_{j=1}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ then $\forall \mu \in M' : \mu \circ \rho_{s_1} \circ \ldots \circ \rho_{s_g} \circ \rho_{t_1} \circ \ldots \circ \rho_{t_h} \in M$, with $\{s_1, \ldots, s_g\} = R_{|\mathcal{E}| \leq |\mathcal{U}|}$ and $\{t_1, \ldots, t_h\} \in 2^{R_{|\mathcal{E}|=|\mathcal{U}|}}$.

Proof. Let $(\Pi = \Pi_1, \ldots, \Pi_{l+1} = \Pi')$ where Π_{j+1} is obtained from Π_j, with $j \in [1 : l]$, by applying the inversion ρ_j on all $\pi_i \in \Pi_j$ with $i \in \mathcal{U}_j$. Let $(\xi = \xi_1, \ldots, \xi_{l+1} = \xi')$ be the scores of the corresponding median problems and $(M = M_1, \ldots, M_{l+1} = M')$ the corresponding medians.

Since by Proposition 3.3.6 $|\xi_j - \xi_{j+1}| \leq \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ holds for all $j \in [1 : l]$, $|\xi - \xi'| = |\xi_1 - \xi_{l+1}| \leq \sum_{j=1}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ follows.

If $\xi - \xi' = \sum_{j=1}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ then $\xi_j - \xi_{j+1} = \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ must hold for all $j \in [1 : l]$. Therefore, if i) $|\mathcal{E}_j| < |\mathcal{U}_j|$ then any median $\mu \in M_{j+1}$ can be transformed into a median of M_j by the application of ρ_j; ii) in the case of $|\mathcal{E}_j| = |\mathcal{U}_j|$ applying ρ_j on a median $\mu \in M_{j+1}$ as well as not applying ρ_j leads to a median of M_j; iii) and in the case of $|\mathcal{E}_i| > |\mathcal{U}_i|$ then any median $\mu \in M_{j+1}$ is also a median of M_j, i.e. the inversion must not be applied. Therefore,

any median $\mu \in M'$ can be transformed into a median of M by applying all inversions ρ_j with $j \in R_{|\mathcal{E}_j|<|\mathcal{U}_j|}$ and applying all inversions of any subset of inversions ρ_j with $j \in R_{|\mathcal{E}_j|=|\mathcal{U}_j|}$. □

This completes the theoretical foundations of the algorithm TCIP that is presented in the next section. The efficient routines that handle linear nodes are implied by Theorem 3.3.5 and the handling of prime nodes is built on Propositions 3.3.6 and 3.3.7.

3.3.4 Algorithm TCIP

Based on the theoretical results from the last section the algorithm TCIP (Tree Common Interval Preserving) is designed as described in the following. An implementation of algorithm TCIP for $k = 3$ in C++ is freely available. This is because TCIP needs an exact IMP solver for the handling of prime nodes and the currently available exact IMP solvers (Caprara, 2003; Siepel and Moret, 2001) only implement the case $k = 3$. Furthermore, this case is most relevant for the reconstruction of phylogenetic trees. Nevertheless, the general case is presented in the following. Some notes for the case $k = 3$ are interspersed. Also the examples and pseudo code are given for the case $k = 3$.

In the first step of the algorithm, the k-signed strong interval tree is computed. In the second step all inversions are applied on the traces starting from the k input permutations to change all k-signs of linear nodes with linear parents according to Theorem 3.3.5. The third step in algorithm TCIP is — from a run time point of view — the most complex one. After identifying all *prime node components*, i.e. the connected subgraphs induced by the prime nodes in the strong interval tree, the quotient permutations of the prime nodes define oIMPs that have to be solved. For prime nodes with a linear parent the k-sign for the oIMP is known because it is inherited from the linear parent. This is not the case for prime nodes with a prime node parent. Recall that the k-sign of an oIMP defines the signs of an individual element in the quotient permutations of the parent prime node. The k-sign tuple assignment in the prime nodes that leads to a minimal number of inversions in a prime node component is determined with a dynamic programming approach.

In the following three subsections it is explained how the pIMP is solved — while developing the necessary theoretical results — in the case of input permutations leading to strong interval trees that are i) unambiguous and have no prime node, ii) unambiguous and have prime nodes, and iii) ambiguous.

Unambiguous trees without prime nodes

Algorithm TCIP processes all linear nodes which have a linear parent. Suppose that the k-signs of a node I and its parent are $s = (s(1), \ldots, s(k))$, respectively $s' = (s'(1), \ldots, s'(k))$. Theorem 3.3.5 implies which scenarios have to be extended by an inversion of I. TCIP's

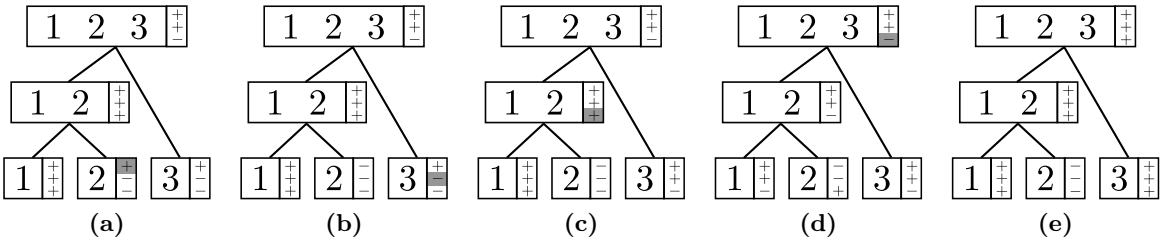


Figure 3.3 – The unambiguous 3-signed strong interval trees arising on a preserving median scenario of Example 3.3.4 with only linear nodes; all nodes are linear and are depicted with rectangular boxes; signs indicate the 3-sign of a node, with $(s(1), s(2), s(3))$ from top to bottom; shaded signs indicate inversions that are applied because of sign differences; a) shows the strong interval tree of the original permutations Π; b)-e) show the corresponding 3-signed strong interval trees after: b) $\{2\}$ is inverted in π_1, c) $\{3\}$ is inverted in π_2, d) $\{1, 2\}$ is inverted in π_3, e) $\{1, 2, 3\}$ is inverted in π_3 leading to $T^3(\Pi, \Pi_\mu)$ with $\Pi_\mu = \{(1\ \text{-}2\ 3)\}^3$

job is to count the number of equal ($s(i) = s'(i)$) and unequal ($s(i) \neq s'(i)$) signs. If less signs are equal than unequal, every i-th scenario with $s(i) = s'(i)$ is extended by inverting I. Otherwise if there are fewer unequal signs than equal signs, every i-th scenario with $s(i) \neq s'(i)$ is extended by inverting I. In case that the number of equal signs and the number of unequal signs is the same one of both possibilities of extending the scenarios has to be chosen. The case of unambiguous trees without prime nodes is illustrated with a small example.

Example 3.3.4. *Let* $\Pi = (\pi_1, \pi_2, \pi_3)$ *with* $\pi_1 = (1\ 2\ 3)$, $\pi_2 = (1\ \text{-}2\ \text{-}3)$, *and* $\pi_3 = (\text{-}3\ 1\ \text{-}2)$ *be the three input permutations. The only non-trivial strong common interval for these permutations is* $C = \{\{1, 2\}\}$. *The 3-signed strong interval tree* $T^3(\Pi)$ *is depicted in Figure 3.3a. There are two differences between the signs of node* $\{2\}$ *and its parent (*$s(\{2\}) = (+, -, -)$ *and* $s(\{1, 2\}) = (+, +, +)$*). This induces the inversion of* $\{2\}$ *in* π_1. *The nodes* $\{1, 2\}$ *and* $\{3\}$ *each have one different sign compared to their parent's 3-sign, leading to the inversions of* $\{1, 2\}$ *in* π_3 *and the inversion of* $\{3\}$ *in* π_2. *The root node has signs* $(+, +, -)$, *so* π_3 *has to be inverted completely. The traces of a parsimonious preserving median scenario with these four inversions and the median* $(1\ \text{-}2\ 3)$ *are:*

1. $\pi_1 = (1\ 2\ 3) \overset{2}{\curvearrowright} (1\ \text{-}2\ 3) = \mu$,
2. $\pi_2 = (1\ \text{-}2\ \text{-}3) \overset{\text{-}3}{\curvearrowright} (1\ \text{-}2\ 3) = \mu$, *and*
3. $\pi_3 = (\text{-}3\ 1\ \text{-}2) \overset{1\text{-}2}{\curvearrowright} (\text{-}3\ 2\ \text{-}1) \overset{\text{-}32\text{-}1}{\curvearrowright} (1\ \text{-}2\ 3) = \mu$

where $\curvearrowright$ *indicates an inversion operation. The effects of the inversions on the permutations* Π *on* $T^3(\Pi, \Pi')$ *are shown in Figures 3.3b to 3.3e.*

If no prime node occurs in the strong interval tree $T^k(\Pi)$, the set of preserving medians as well as the scenarios leading to each of the preserving medians is directly defined by the tree. The following theorem gives the number of preserving medians.

Theorem 3.3.8. *Let Π be a sequence of k permutations such that $T^k(\Pi)$ has only linear nodes. Let a be the number of nodes in $T^k(\Pi)$ for which $|\mathcal{E}| = |\mathcal{U}|$. Then there exist 2^a preserving medians for Π.*

Proof. All inversions implied by linear nodes commute by definition as they invert complete nodes of $T^k(\Pi)$, which do commute. Let $I_1, \ldots, I_a$ be the set of nodes with $|\mathcal{E}| = |\mathcal{U}|$. Every node I_i, with $i \in [1 : a]$, has two possible options how to extend the traces. As inversions implied by linear nodes commute, those options can be freely combined. This leads to 2^a possibilities. All other nodes of the tree imply a unique set of inversions to be applied to the traces. Therefore, the number of medians is 2^a. □

If only one preserving median is of interest just one of the two possibilities has to be chosen in case $|\mathcal{E}| = |\mathcal{U}|$. In case of $k = 3$ there is always a unique median if the strong interval tree has no prime nodes. This is covered by the following corollary.

Corollary 3.3.9. *Let Π be a sequence of k permutations such that $T^k(\Pi)$ has only linear nodes. If k is odd then Π has a unique median.*

Unambiguous trees with prime nodes

In this subsection it is assumed that each prime node of the strong interval tree $T^k(\Pi)$ has a linear parent.

Let Π be a sequence of k permutations where $T^k(\Pi)$ has a prime node I and μ be a permutation that is preserving with respect to Π. A preserving scenario for Π to μ transforms $T^k(\Pi)$ into $T^k(\Pi, \Pi_\mu)$, where $\Pi_\mu = \{\mu\}^k$. In $T^k(\Pi, \Pi_\mu)$ the children of I are in the same order in all k permutations, i.e. I is a linear node.

Therefore, for the purpose of solving the preserving median problem, the arrangement of the child nodes must be transformed into a linear order, i.e. node I is transformed into a linear node. This must happen such that the number of preserving inversions, necessary to obtain the linear order, plus the number of additional inversions, implied by k-sign differences between node I and its parent and between node I and its the children is minimal.

The key for finding a preserving median scenario for a prime node is the fact that every inversion, being a union of children of a prime node, is preserving (see Proposition 3.3.1). That is the order and orientation of the children of a prime node can freely be rearranged. As formally described in Proposition 3.3.3, each inversion being a union of children of a prime node can be mimicked by a corresponding quotient inversion of the corresponding signed

quotient permutations. Therefore, solving the oIMP for the signed quotient permutations induced by a prime node and some k-sign s gives a preserving scenario for the children of a prime node. This scenario will transform $T^k(\Pi)$ into $T^k(\Pi, \Pi')$, such that the children of the prime node have the same increasing or decreasing order, given by a median of the oIMP solution. That is the node is linear and has k-sign s. The sign differences between s and the k-sign of the linear parent induce some additional inversions that have to be regarded. In Theorem 3.3.11 it is shown that solving the oIMP for the signed quotient permutations and the k-sign inherited from the linear parent leads to a minimal number of preserving inversions, i.e. a preserving median scenario. For this the following corollary is necessary. It applies the result for the IMP from Proposition 3.3.7 to the oIMP.

Corollary 3.3.10. *Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a sequence of signed permutations and I a prime node of $T^k(\Pi)$ with parent I_0 and child nodes $(I_1, \ldots, I_l)$. Let $(s_0, \ldots, s_l)$ and $(s'_0, \ldots, s'_l)$ be two sequences of k-signs. Then $\Pi_{|I,s_1,\ldots,s_l}$ and the k-sign s_0 define an oIMP with score ξ and set of medians M. $\Pi_{|I,s'_1,\ldots,s'_l}$ and the k-sign s'_0 define another oIMP with score ξ' and set of medians M'. Let $\mathcal{U}_j$ denote $\mathcal{U}(s_j, s'_j)$ and $\mathcal{E}_j$ denote $\mathcal{E}(s_j, s'_j)$ for $j \in [0 : l]$. Let $\sigma_{|I,t_1,\ldots,t_l}$ denote a signed quotient permutation of node I where a sign $t_i \in [1 : l]$ gives the sign of element i in σ.*

It holds that $|\xi - \xi'| \leq \sum_{j=0}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$. If $\xi - \xi' = \sum_{j=0}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ then it holds that $\forall t_0 \circ \mu'_{|I,t_1,\ldots,t_l} \in M' : t'_0 \circ \mu_{|I,t'_1,\ldots,t'_l} \in M$ where

$$t'_i = \begin{cases} t_i & \text{if } |\mathcal{U}_i| < |\mathcal{E}_i|, \\ -t_i & \text{if } |\mathcal{E}_i| < |\mathcal{U}_i|, \text{ and} \\ \text{either } t_i \text{ or } -t_i & \text{if } |\mathcal{U}_i| = |\mathcal{E}_i|. \end{cases}$$

Proof. Let ρ_0 be the inversion of a complete permutation and ρ_j be the inversion of element j, with $j \in [1 : l]$, in a signed quotient permutation. Clearly, $\sum_{i=1}^{k} d(\pi_i, s_0(i) \circ \mu) = \sum_{i=1}^{k} d(s_0(i) \circ \pi_i, \mu)$. Therefore, the IMPs given by $(s_0(1) \circ \pi_{1|I,s_1(1),\ldots,s_l(1)}, \ldots, s_0(k) \circ \pi_{k|I,s_1(k),\ldots,s_l(k)})$ and $(s'_0(1) \circ \pi_{1|I,s'_1(1),\ldots,s'_l(1)}, \ldots, s'_0(k) \circ \pi_{k|I,s'_1(k),\ldots,s'_l(k)})$ can be considered instead of the corresponding oIMPs.

The signed permutations $s'_0(i) \circ \pi_{i|I,s'_1(i),\ldots,s'_l(i)}$ can be obtained from $s_0(i) \circ \pi_{i|I,s_1(i),\ldots,s_l(i)}$, with $i \in [1 : k]$, by applying the inversions in $\{\rho_j : i \in \mathcal{U}_j, j \in [1 : l]\}$. Therefore, Proposition 3.3.7 can be applied and $|\xi - \xi'| \leq \sum_{j=0}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$ holds. In the case of maximal score difference the construction of medians in M from medians in M' with inversions ρ_i translates directly to sign differences between t_i and t'_i. □

Theorem 3.3.11. *Let $T^k(\Pi)$ be a k-signed strong interval tree and I be a prime node of $T^k(\Pi)$ that has a linear parent node and l linear child nodes $(I_1, \ldots, I_l)$. The k-sign of the linear parent node of I is denoted by s_0 and the k-signs of the linear child nodes I_i, with*

$i \in [1:l]$, are denoted by s_i. Every inversion in a preserving median scenario of Π being a union of children of I corresponds to a quotient inversion of a median scenario of the oIMP defined by $\Pi_{|I,s_1,\ldots,s_l}$ and the k-sign s_0.

Proof. By Proposition 3.3.1 the inversions that are a union of children of a prime node are preserving. Therefore, it remains to show that the inversion scenarios corresponding to the quotient inversion scenarios of the oIMP are parsimonious and that all parsimonious inversion scenarios are implied by the oIMP.

Let s_0 be the inherited k-sign of the linear parent of prime node I and $s_1, \ldots, s_l$ the k-signs of the l linear child nodes. Note that I has no prime child nodes. The signed quotient permutations $\Pi_{|I,s_1,\ldots,s_l}$ of node I and k-sign s_0 define an oIMP. Let ξ be the score and M be the set of medians of this oIMP. The signed permutations Π are transformed into $\hat{\Pi}$ by the application of the ξ inversions on Π, that correspond to the quotient inversions of an oriented median scenario from $\Pi_{|I,s_1,\ldots,s_l}$ to $s_0 \circ \mu$. Let $\hat{s_i}$, with $i \in [0:l]$, denote the k-signs of node I and its l child nodes in $T^k(\Pi, \hat{\Pi})$. The quotient permutations of node I in $T^k(\Pi, \hat{\Pi})$ are either μ or $-\mu$ depending on s_0. That is node I is linear and has k-sign $\hat{s_0} = s_0$. Therefore, no additional inversions of I are necessary by Theorem 3.3.5. Furthermore, $\hat{s_i} = \hat{s_0}$ holds for all $i \in [1:l]$ if element i is positive in μ and $\hat{s_i} = -\hat{s_0}$ if element i is negative in μ. Therefore, no additional inversion of a child of I is necessary by Theorem 3.3.5.

Let $(s'_0, s'_1, \ldots, s'_l) \neq (s_0, s_1, \ldots, s_l)$ be k-signs. Let $\mathcal{U}_i = \mathcal{U}(s_i, s'_i)$ and $\mathcal{E}_i = \mathcal{E}(s_i, s'_i)$, with $i \in [1:l]$. Let ξ' be the score and M' be the set of medians of the oIMP defined by $\Pi_{|I,s'_1,\ldots,s'_l}$ and s'_0. The application of the inversions on Π corresponding to the quotient inversions of an oriented median scenario from $\Pi_{|I,s'_1,\ldots,s'_l}$ to $\mu' \in M'$ results in the k-signed strong interval tree $T^k(\Pi, \hat{\Pi}')$. Node I is linear in $T^k(\Pi, \hat{\Pi}')$, too but has k-sign s'_0. The oIMP scenario that was computed for $\Pi_{|I,s'_1,\ldots,s'_l}$ is applied to the actual quotient permutation $\Pi_{|I,s_1,\ldots,s_l}$, this results in differences between the k-signs of the child nodes and node I in $T^k(\Pi, \hat{\Pi}')$. That is k-sign of child node I_i is given by $\hat{s_i}'(j) = -s'_0(j)$ for $j \in \mathcal{U}_i$ and $\hat{s_i}'(j) = s'_0(j)$ for $j \in \mathcal{E}_i$, $i \in [1:l]$. Thus, by Theorem 3.3.5, exactly $\sum_{i=0}^{l} \min(|\mathcal{U}_i|, |\mathcal{E}_i|)$ additional inversions are necessary due to sign differences of node I and its parent node, respectively its child nodes. Therefore, the solution given by the oIMP with different k-signs $(s'_0, \ldots, s'_l)$ can not provide a better solution as, by Corollary 3.3.10, $|\xi - \xi'| \leq \sum_{j=0}^{l} \min(|\mathcal{U}_j|, |\mathcal{E}_j|)$.

However, the use of $(s'_0, \ldots, s'_l)$ could lead to other preserving medians for $\xi' < \xi$, with $\xi - \xi' = \sum_{i=0}^{l} \min(|\mathcal{U}_i|, |\mathcal{E}_i|)$. In the following it is shown that this is not the case.

Consider again the differences between the k-signs of node I and its child nodes and its parent in $T^k(\Pi, \hat{\Pi}')$ that are given above. Let ρ_0 be the inversion of I and ρ_i be the inversion of I_i, with $i \in [1:l]$. Furthermore, the corresponding quotient inversions are given by $\rho_{0|I} = \{1, \ldots, l\}$ and $\rho_{i|I} = \{i\}$, with $i \in [1:l]$. By Theorem 3.3.5 the inversion ρ_i in the j-th scenario belongs to a preserving median scenario, with $j \in \mathcal{U}_i$ if $|\mathcal{U}_i| \leq |\mathcal{E}_i|$ or $j \in \mathcal{E}_i$ if

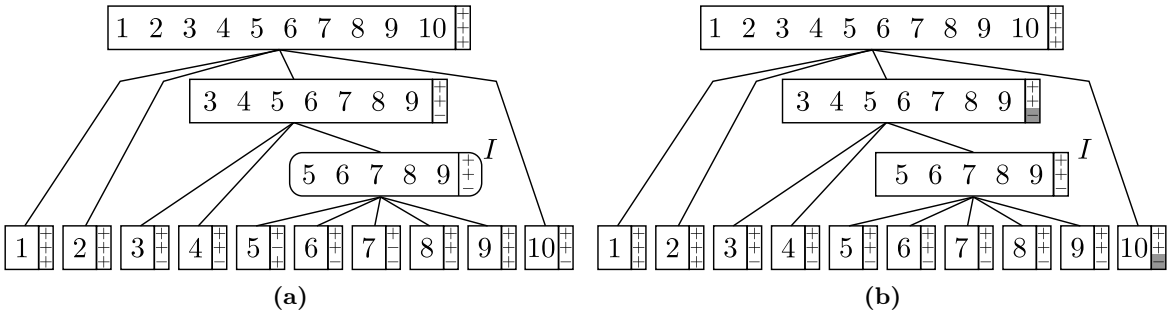


Figure 3.4 – a) Unambiguous strong interval tree of Example 3.3.5 with one prime node; b) Unambiguous strong interval $T^3(\Pi, \Pi')$ resulting from the application of the eight inversions implied by the median (1 2 3 4 5) of the oIMP as shown in Example 3.3.5; linear nodes are depicted with rectangular boxes, prime nodes are depicted with round boxes; signs indicate the 3-sign of a node, with (s_1, s_2, s_3) from top to bottom; shaded signs indicate inversions that have to be applied because of sign differences between a linear node and its parent

$|\mathcal{E}_i| \leq |\mathcal{U}_i|$. The application of the corresponding quotient permutations to a median $\mu' \in M'$ leads by Corollary 3.3.10 to a median $\mu \in M$. □

Hence, solving the oIMP for a prime node, which is defined by the signed quotient permutations and the k-sign of the linear parent, gives a preserving median scenario. Furthermore, no different preserving median scenario can be found when different k-signs are used for the quotient permutations or the oIMP. This justifies why a prime node inherits the k-sign from its linear parent.

Example 3.3.5. *Let* π_1 = (1 2 3 4 5 6 7 8 9 10), π_2 = (1 2 3 4 8 -5 -7 9 6 10), *and* π_3 = (1 2 5 -6 -7 -8 9 -4 -3 -10) *be the three input permutations. The non-trivial strong common intervals for* $\{\pi_1, \pi_2, \pi_3\}$ *are* $C = \{\{3,4,5,6,7,8,9\}, \{5,6,7,8,9\}\}$. *The* 3*-signed strong interval tree is depicted in Figure 3.4a. The only prime node* I *inherits the* 3*-sign* $(+,+,-)$ *from the linear parent and the three signed quotient permutations are* $\pi_{1|I}$ = (1 2 3 4 5), $\pi_{2|I}$ = (4 -1 -3 5 2), *and* $\pi_{3|I}$ = (1 -2 -3 -4 5). *Solving the oIMP for* 3*-sign (+,+,-), by solving the IMP for* $\pi_{1|I}, \pi_{2|I}$, *and* $-\pi_{3|I}$, *leads to a median* μ_I = (1 2 3 4 5) *with a score of eight. Note, there are* 11 *other medians for the oIMP:* (-5 -4 3 -2 -1), (-5 -3 -2 4 -1), (-5 4 -3 2 -1), (-5 4 3 -2 -1), (1 -4 -3 -5 2), (1 -4 -3 -2 -5), (1 -4 -3 -2 5), (1 -2 3 4 5), (1 2 -4 -3 5), (4 -1 -3 -2 -5), *and* (4 -1 -3 -2 5). *One parsimonious scenario for this oIMP to the median* $\mu_{|I}$ *is defined by the following scenarios of quotient inversions.*

- $\pi_{1|I} = \mu_{|I}$
- $\pi_{2|I} \overset{5}{\curvearrowright}$ (4 -1 -3 -5 2) $\overset{4\text{-}1}{\curvearrowright}$ (1 -4 -3 -5 2) $\overset{\text{-}52}{\curvearrowright}$ (1 -4 -3 -2 5) $\overset{\text{-}4\text{-}3\text{-}2}{\curvearrowright}$ $\mu_{|I}$

- $\pi_{3|I} \overset{\text{-2}}{\curvearrowright} (1\ 2\ \text{-3}\ \text{-4}\ 5) \overset{\text{-3}}{\curvearrowright} (1\ 2\ 3\ \text{-4}\ 5) \overset{\text{-4}}{\curvearrowright} (1\ 2\ 3\ 4\ 5) \overset{12345}{\curvearrowright} = -\mu_{|I}$

The quotient inversions in the oIMP scenario directly correspond to inversions in the pIMP scenario. If there is no interest in a scenario, the order of the intervals can be permuted into the order given by an oIMP median directly. The corresponding preserving inversion scenario for Π *is given by the following traces:*

- *no inversions have to be applied to* $\pi_1 = \pi'_1$ *as* $\pi_{1|I} = \mu_{|I}$
- $\pi_2 \overset{9}{\curvearrowright} (1\ 2\ 3\ 4\ 8\ \text{-5}\ \text{-7}\ \text{-9}\ 6\ 10) \overset{\text{8-5}}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ \text{-8}\ \text{-7}\ \text{-9}\ 6\ 10) \overset{\text{-96}}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ \text{-8}\ \text{-7}\ \text{-6}\ 9\ 10) \overset{\text{-8-7-6}}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10) = \pi'_2$
- $\pi_3 \overset{\text{-6}}{\curvearrowright}\overset{\text{-7}}{\curvearrowright}\overset{\text{-8}}{\curvearrowright} \overset{56789}{\curvearrowright} (1\ 2\ \text{-9}\ \text{-8}\ \text{-7}\ \text{-6}\ \text{-5}\ \text{-4}\ \text{-3}\ \text{-10}) = \pi'_3$

After these eight inversions, Π *is transformed into* $\Pi' = (\pi'_1, \pi'_2, \pi'_3)$*. The* 3*-signed strong interval tree* $T^3(\Pi, \Pi')$ *is shown in Figure 3.4b. Node* I *is linear and the* 3*-sign of all nodes except for nodes* $\{10\}$ *and* $\{3, 4, 5, 6, 7, 8, 9\}$ *are equal to the* 3*-sign of their linear parents. Therefore, the third trace must be extended in the following way:*

- $\pi'_3 \overset{\text{-9-8-7-6-5-4-3}}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ \text{9-10}) \overset{\text{-10}}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10) = \mu$

The permutations π'_1 *and* π'_2 *are already equal to the preserving median. There are* 11 *more preserving medians for this pIMP which can be found by using the other medians of the oIMP. The preserving median scenario has a score of ten.*

The use of different k-signs for the quotient permutations and the definition of the oIMP for a prime node is exemplified in the following.

Example 3.3.6. *The effect of using different* k*-signs is exemplified here for the prime node* I *in the* 3*-signed strong interval tree* $T^3(\Pi)$ *(see Figure 3.4a) of the permutations* Π *as given in Example 3.3.5. Consider the* 3*-signs* $s'_1 = (+,-,+)$*,* $s'_2 = (+,+,+)$*,* $s'_3 = (+,-,+)$*,* $s'_4 = (+,+,+)$*, and* $s'_5 = (+,-,+)$*. The signed quotient permutations* $\Pi_{|I,s'_1,\ldots,s'_5}$ *are given by* $\pi_{1|I,s'_1,\ldots,s'_5} = (1\ 2\ 3\ 4\ 5)$*,* $\pi_{2|I,s'_1,\ldots,s'_5} = (4\ \text{-1}\ \text{-3}\ \text{-5}\ 2)$*,* $\pi_{3|I,s'_1,\ldots,s'_5} = (1\ 2\ 3\ 4\ 5)$*. The sign for the oIMP is given by* $s' = (+,+,+)$*.*

The differences between the 3*-signs of the parent and child nodes of* I *(compare Example 3.3.5) and the* 3*-signs considered here are given by:* $\mathcal{U}(s, s') = \{3\}$*,* $\mathcal{U}(s_1, s'_1) = \emptyset$*,* $\mathcal{U}(s_2, s'_2) = \mathcal{U}(s_3, s'_3) = \mathcal{U}(s_4, s'_4) = \{3\}$*,* $\mathcal{U}(s_5, s'_5) = \{2\}$*. By Corollary 3.3.10 the maximal possible difference resulting from the use of the different* 3*-signs is* $\min(|\mathcal{U}(s, s')|, |\mathcal{E}(s, s')|) + \sum_{i=1}^{5} \min(|\mathcal{U}(s_i, s'_i)|, |\mathcal{E}(s_i, s'_i)|) = 5$*.*

Solving the oIMP defined by $\Pi_{|I,s'_1,\ldots,s'_5}$ *and the* k*-sign* s' *results in a single median* $\mu'_{|I} = (1\ 2\ 3\ 4\ 5)$ *with score* $\xi' = 3$*. Thus, the maximal score difference is realised with these* 3*-signs.*

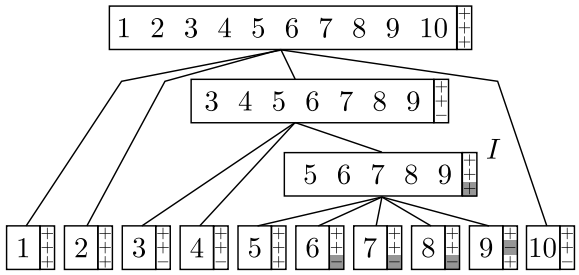


Figure 3.5 – Strong interval tree $T^3(\Pi, \hat{\Pi}')$ corresponding to the permutations $\hat{\Pi}'$ that are generated from Π by applying the inversions of the median scenario of the oIMP with different 3-signs as given in Example 3.3.6; linear nodes are depicted with rectangular boxes, prime nodes are depicted with round boxes; signs indicate the 3-sign of a node, with (s_1, s_2, s_3) from top to bottom; shaded signs indicate the additional inversions necessary for node I and its child nodes that have to be applied because of the use of different 3-signs for the oIMP

A scenario for this median is given by the quotient inversions of the sets of elements $\{4, 1\}$, $\{5, 2\}$, *and* $\{4, 3, 2\}$ *in* $\pi_{2|I,s'_1,\ldots,s'_5}$. *Applying those quotient inversions on the actual quotient permutations* $\Pi_{|I}$ *results in:*

- $\pi_{1|I} = (1\ 2\ 3\ 4\ 5)$
- $\pi_{2|I} = (4\ \text{-}1\ \text{-}3\ 5\ 2) \overset{4\text{-}1}{\curvearrowright} (1\ \text{-}4\ \text{-}3\ 5\ 2) \overset{52}{\curvearrowright} (1\ \text{-}4\ \text{-}3\ \text{-}2\ \text{-}5) \overset{\text{-}4\text{-}3\text{-}2}{\curvearrowright} = (1\ 2\ 3\ 4\ \text{-}5)$
- $\pi_{3|I} = (1\ \text{-}2\ \text{-}3\ \text{-}4\ 5)$

Observe that the application of the scenario from $\Pi_{|I,s'_1,\ldots,s'_5}$ *to* $\mu'_{|I}$ *on the signed quotient permutations* $\Pi_{|I,s_1,\ldots,s_5}$ *results in three signed permutations with a linear order but differences in the signs of the elements. Applying the corresponding preserving inversions to the permutations in* Π *results in* $\hat{\Pi}' = (\hat{\pi}'_1, \hat{\pi}'_2, \hat{\pi}'_3)$ *with :*

- $\hat{\pi}'_1 = \pi_1$
- $\pi_2 \overset{8\text{-}5}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ \text{-}8\ \text{-}7\ 9\ 6\ 10) \overset{96}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ \text{-}8\ \text{-}7\ \text{-}6\ \text{-}9\ 10) \overset{\text{-}8\text{-}7\text{-}6}{\curvearrowright} (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ \text{-}9\ 10) = \hat{\pi}'_2$
- $\hat{\pi}'_3 = \pi_3$

The 3*-signed strong interval tree of the resulting permutations is depicted in Figure 3.5. There are four child nodes of* I *that have a different* 3*-sign compared to the* 3*-sign of* I*, furthermore the* 3*-sign of* I *is different from the* 3*-sign of its parent. These differences imply the inversion of the elements* 6, 7, 8 *and the set* $\{5, \ldots, 9\}$ *in* π_3 *and the inversion of element* 9 *in* π_2. *Applying these inversions leads to the strong interval tree as depicted in Figure 3.4b. Observe that these inversions correspond to the quotient inversions of the elements* 2, 3, 4

and the set $\{1, \ldots, 5\}$ *in* $\pi_{3|I}$ *and the quotient inversion of element* 5 *in* $\pi_{2|I}$ *included in the oIMP scenario computed for the original signs as given in Example 3.3.5 but missing in the oIMP scenario defined by the different signs. Thus, the five additional inversions necessary to equalise the signs of node* I *and its parent and child nodes compensate the five inversions that have been saved by solving the oIMP with the different* k*-signs. Furthermore, the application of the five additional inversions leads to the same result as if the original signs are used for solving the oIMP.*

TCIP does not compute the traces connecting the quotient permutations with their medians, but these can be easily computed with known algorithms (e.g. Tannier et al., 2007). Note that there may be a huge number of parsimonious scenarios which could be computed with the methods given in Braga et al. (2008); Siepel (2001).

Solving the pIMP with separate prime nodes can be accomplished by solving one oIMP for each prime node. The size of the quotient permutations in a prime node is always smaller than or equal to the original permutation size. This may lead to a computation time reduction compared to solving the original IMP. It will be shown empirically in Section 3.4 that the size of the quotient permutations in prime nodes is usually much smaller than the original permutation size.

Ambiguous trees

Similar to preserving sorting scenarios (compare Bérard et al., 2007; Diekmann et al., 2007), the most difficult case for the pIMP occurs when there are connected prime nodes in the strong interval tree $T^k(\Pi)$. Ambiguous trees contain subtrees consisting of connected prime nodes. In the following it is shown that each of these subtrees can be handled separately and methods for solving such a subtree of connected prime nodes are presented. First a brute force method is explained, afterwards a more efficient recursive procedure is introduced which is further improved at the end of this section.

Ambiguous trees have prime nodes with prime nodes as parents. For such prime nodes and prime root nodes the k-sign is unknown. In this case the sign vector of the corresponding oIMP is not known. Furthermore, the sign of elements in the signed quotient permutation of a prime node corresponding to prime child nodes is unknown. This is because the k-sign of a prime node child — which is unknown — defines the signs of the corresponding element of the quotient permutations in the parent node. Assigning a k-sign to each of the prime nodes in the component determines the input k-sign for each of the oIMPs and the unknown k-signs in the signed quotient permutations. In order to identify the inversions of unions of children of the prime nodes that belong to a preserving median scenario, a k-sign assignment has to be found such that the sum of the scores of the oIMPs is minimal.

A simple brute force method solving this problem tests all possible k-sign assignments to the prime nodes in order to find an assignment with a minimal sum of the scores of the oIMPs. There are $2^{k(p-1)}$ possible k-sign assignments to the p prime nodes of a prime node component because the sign for the root node of the connected component is determined. For each assignment p oIMPs have to be solved, which are NP-hard. Therefore, this approach will be intractable even for moderately large prime node components. Observe that many of the subproblems (oIMPs) solved in the brute force approach overlap. That is the same oIMPs are solved over and over. This can be improved by adopting a dynamic programming approach.

In order to compute a scenario with a minimum score for a connected prime node subtree, the induced median problems within the subtree are solved bottom up. Let I be a prime node of the strong interval tree $T^k(\Pi)$, $I_1, \ldots, I_p$ be the p prime child nodes of I, and $I_{p+1}, \ldots, I_{p+l}$ be the l linear child nodes. Note that $p = 0$ or $l = 0$ is possible, as long as $p+l > 0$ holds. Let $s, s_1, \ldots, s_p$ be k-signs in $\Sigma = \{+,-\}^k$. Each linear child node I_{p+j}, with $j \in [1:l]$, has a k-sign, denoted by s_{p+j}. The score of the solution of the oIMP defined by the input k-sign s and the k signed quotient permutations $\pi_{i|I,s_1,\ldots,s_{p+l}}$, $i \in [1:k]$, of I and the k-signs $s_1, \ldots, s_{p+l}$, is denoted by $\xi(I, s, s_1, \ldots, s_{p+l})$. Recall, the elements of the signed quotient permutations $\pi_{i|I,s_1,\ldots,s_{p+l}}$ that are the input for the oIMP have signs according to $s_1, \ldots, s_{p+l}$. Let

$$\xi_{\min}(I, s) = \min_{(s_1,\ldots,s_p)\in\Sigma^p} \left\{ \xi(I, s, s_1, \ldots, s_{p+l}) + \sum_{i=1}^{p} \xi_{\min}(I_i, s_i) \right\} \tag{3.1}$$

be the recursive definition of the score of the prime node component with root node I having sign s, i.e. the minimal number of quotient inversions needed to solve all oIMPs induced by the prime node component with root node I having sign s. These quotient inversions correspond to preserving inversions of Π that belong to a preserving inversion median scenario. For a prime node component with root node I having a linear parent node that inherits its k-sign s, the minimal number of inversions is given by $\xi_{min}(I, s)$. If the root node of a prime node component is the root of $T^k(\Pi)$, the minimal number of inversions is given by $\min\{\xi_{\min}(I, \{+\}^k), \xi_{\min}(I, \{-\}^k)\ \}$. If a prime node I has no prime child node, $\xi_{\min}(I, s)$ is just the score of the induced oIMP where the signs of all elements in the input permutations are all determined by the linear child nodes.

In order to show that the k-sign of a linear parent node may be inherited by the root node of a prime node component, a slight modification of Theorem 3.3.11 is needed for the case of ambiguous strong interval trees.

Theorem 3.3.12. *Let $T^k(\Pi)$ be a k-signed strong interval tree and I be a prime node of $T^k(\Pi)$ with l linear and p prime child nodes, with $l \geq 0$ and $p \geq 0$ such that $l + p > 0$. Let $(I_1, \ldots I_p)$ denote the possibly empty list of prime child nodes and $(I_{p+1}, \ldots, I_{p+l})$ denote the*

possibly empty list of linear child nodes of I. Let s_0 denote the k sign of I that is inherited from its linear parent if existent or that is assigned to node I and let s_i, with $1 \leq i \leq p$, denote the k-signs assigned to prime child nodes I_i, and s_i, with $p < i \leq p + l$, the k-signs of the linear child nodes of I.

Every inversion in a preserving median scenario of Π which is a union of children of I corresponds to a quotient inversion of an oriented median scenario of an oIMP defined by $\Pi_{|I,s_1,\ldots,s_{p+l}}$ and the k-sign s_0 where the k-signs $s_0, \ldots, s_{p+l}$ belong to a k-sign assignment, to the prime nodes of the connected prime node component including I, with minimal score.

Proof. If the sign assignment does not lead to a minimal score, S cannot be a preserving median scenario. Thus, in the following a minimal k-sign assignment is assumed.

Let Π be a sequence of permutations and I be a prime node of $T^k(\Pi)$. Consider an assignment of k-signs to the prime nodes of the connected prime node component containing I. Let $s_1, \ldots, s_p$ be the k-signs assigned to the prime child nodes of i, $s_{p+1}, \ldots, s_{p+l}$ be the k-signs of the linear children, and s_0 be the k-sign of the linear parent if I has a linear parent. Otherwise s_0 denotes the k-sign assigned to node I. Let S denote a preserving scenario for Π where the inversions that correspond to quotient inversions of I are given by solving the oIMP for $\Pi_{I|s_1,\ldots,s_{p+l}}$ with k-sign s_0. The inversions in S corresponding to quotient inversions of node I commute with every other inversion of S. Therefore, it can be assumed that the inversions in S that correspond to quotient inversions of node I are the last inversions in each of the k scenarios. Now, let Π' be the result of applying all inversions of S except the inversions corresponding to quotient inversions of node I. Theorem 3.3.11 can be applied because I is the only prime node in $T^k(\Pi, \Pi')$ and consequently all neighbours of I are linear nodes.

□

In the following two propositions which are useful for improving the run time of the first version of algorithm TCIP are shown.

Proposition 3.3.13. *It is sufficient to use a set $\Sigma' \subset \{+,-\}^k$ with the property*

$$\forall s \in \{+,-\}^k : (s \in \Sigma' \Leftrightarrow -s \notin \Sigma')$$

in Equation (3.1) *in order to compute $\xi_{\min}(I, s)$.*

Proof. Let $(s_0, s_1, \ldots, s_p)$ and $(s'_0, s'_1, \ldots, s'_p)$ be two tuples of k-signs with $s_i = s'_i$ or $s_i = -s'_i$. Let $\mathcal{E}_i = \mathcal{E}(s_i, s'_i)$ and $\mathcal{U}_i = \mathcal{U}(s_i, s'_i)$. The proof has three parts. In the first part, it will be shown that for any node I of the strong interval tree with p prime child nodes and l linear child nodes $\xi(I, s_0, s_1, \ldots, s_p, s_{p+1}, \ldots, s_{p+l}) = \xi(I, s'_0, s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})$ holds. In the second part, it will be shown that for any node of the strong interval tree $\xi_{\min}(I, s)$=$\xi_{\min}(I, -s)$ holds. In the third part, these results are used to show the proposition.

i) The computation of $\xi(I, s_0, s_1, \ldots, s_{p+l})$ is done by solving the oIMP defined by the k signed quotient permutations of I and the input k-sign s_0. The sign of the elements of the k quotient permutations are determined by the k-signs $(s_1, \ldots, s_p, \ldots, s_{p+l})$. Similarly $\xi(I, s'_0, s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})$ is computed by solving the oIMP defined by the k signed quotient permutations of I and the k-sign s'_0 where the sign of the elements of the k quotient permutations are determined by the k-signs $(s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})$. In case of $s_i = s'_i$, $i \in [1:p]$ it holds that $\mathcal{E}_i = \{1, \ldots, k\}$ and $\mathcal{U}_i = \emptyset$. For $s_i = -s'_i$, $\mathcal{E}_i = \emptyset$ and $\mathcal{U}_i = \{1, \ldots, k\}$ holds. Therefore, by Corollary 3.3.10, $|\xi(I, s_0, s_1, \ldots, s_p, \ldots, s_{p+l}) - \xi(I, s'_0, s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})| \leq \sum_{i=0}^{p} \min(|\mathcal{U}_i|, |\mathcal{E}_i|) = 0$. That is

$$\xi(I, s_0, s_1, \ldots, s_p, s_{p+1}, \ldots, s_{p+l}) = \xi(I, s'_0, s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})$$

holds for any node I.

ii) Comparing $\xi_{\min}(I, s)$ and $\xi_{\min}(I, -s)$ reduces to the comparison of $\xi(I, s, s_1, \ldots, s_{p+l})$ and $\xi(I, -s, s_1, \ldots, s_{p+l})$. The arguments from part i) of the proof show that these two scores of the oIMPs with input k-sign s and $-s$ are equal. Therefore, for any node I it holds that $\xi_{\min}(I, s) = \xi_{\min}(I, -s)$.

iii) Let $(s_1, \ldots, s_p)$ be a tuple of signs in Equation (3.1) for which $\xi_{\min}(I, s)$ is minimal, and $\Sigma' \subset \{+, -\}^k$ a set of k-signs where $\forall s \in \{+, -\}^k : (s \in \Sigma' \Leftrightarrow -s \notin \Sigma')$ holds. As any sign s_i can be replaced by sign $-s_i$ in Equation (3.1), any k-sign $s_i \notin \Sigma'$ can replaced by $-s_i$, which is in Σ'. Hence, there exists a tuple of signs $(s'_1, \ldots, s'_p)$ that leads to a minimal value for $\xi_{\min}(I, s)$ and $s'_i \in \Sigma'$.

Therefore, the set of k-signs to be considered can be restricted to Σ'.

Finally it is shown that all medians are found when Σ' is used. This happens in two steps, first the differences for the oIMP are given and second it is shown that the differences in the medians of neighbouring prime nodes annihilate each other.

i) Let $t_0 \circ \mu_{|I, t_0, \ldots, t_p, t_{p+1}, t_{p+l}}$ be a median of the oIMP defined by the k-sign s_0 and the k quotient permutations of I signed according to $(s_1, \ldots, s_p, \ldots, s_{p+l})$. Then, by Corollary 3.3.10, the oIMP defined by the k quotient permutations of I signed according to $(s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l})$ and the k-sign s'_0 has a median $t'_0 \circ \mu_{|I, t'_0, \ldots, t'_p, t_{p+1}, t_{p+l}}$ with $t'_i = -t_i$ iff $\mathcal{E}_i = \emptyset$, i.e. $|\mathcal{E}_i| < |\mathcal{U}_i|$ and $t'_i = t_i$ iff $\mathcal{U}_i = \emptyset$, i.e. $|\mathcal{U}_i| < |\mathcal{E}_i|$.

ii) Let J be a prime child node of I and let j, with $0 \leq j \leq p$, be the index in the k-sign vector of node I that corresponds to node J. Changing k-sign s_j to s'_j for solving the oIMP in node I corresponds to changing the input k-sign of the oIMP for node J. Then solving the oIMP for node J with k-sign s'_j instead of s_j results in an inverted iff $t'_j \neq t_j$ in I. Hence, the differences annihilate each other.

□

The above argumentation for the equality of the median sets while only a k-sign set Σ' is used may alternatively be regarded as follows. A k-sign expresses the differences in the orientation of a node or of the signs of an element. In the currently used notation they are defined absolute by referring to π_1. Alternatively, the k-signs can be defined such that they specify relative differences. Ergo a different k-sign tells that there is a difference in the orientation or sign, but it does not tell the actual orientations. There is no difference between k-signs s and $-s$ in this formalism.

There are many possible choices for a set Σ' satisfying the property stated in Proposition 3.3.13. The following corollary gives one simple representative of the possible choices. This representative is used in the implementation.

Corollary 3.3.14. *It is sufficient to use the set* $\Sigma' = \{s : s \in \{+,-\}^k, s(1) = +\}$ *in Equation* (3.1) *in order to compute* $\xi_{\min}(I, s)$.

This set is chosen for the following reason. The application of the inversions on Π that are implied by the inversions on the prime permutations given by a minimal solution of Equation (3.1) transforms the prime nodes of the prime node component into linear nodes. These linear nodes will have k-signs with a + at the first position. This is compliant with the numbering scheme used for the computation of the quotient permutations that leads to k-signs with a + at the first position in all linear nodes.

This halves the number of k signs that have to be considered in Equation (3.1). The number of oIMPs to be solved is analysed in detail in Section 3.3.5.

Proposition 3.3.15. *For determining* $\xi_{\min}(I, s)$ *in Equation* (3.1)*, it is sufficient to consider those tuples* $(s_1, \ldots, s_p) \in \Sigma^p$ *with the property that for each* k*-sign* s_i*,* $i \in [1 : p]$*, exists no other* k*-sign* s_i' *with* $\xi_{\min}(I_i, s_i') < \xi_{\min}(I_i, s_i)$.

Proof. i) Let I be a prime node with p prime child nodes and l linear child nodes and $s \neq s'$ be two k-signs. Let $(s_1, \ldots, s_p) \in \Sigma^p$ be a tuple of k-signs that is used to find the minimum value for computing $\xi_{\min}(I, s)$, respectively $\xi_{\min}(I, s')$. Due to the fact that s, respectively s', has no influence on the recursive part of Equation (3.1) it follows that $|\xi_{min}(I, s) - \xi_{min}(I, s')| = |\xi(I, s, s_1, \ldots, s_{p+l}) - \xi(I, s', s_1, \ldots, s_{p+l})|$. Thus, by Corollary 3.3.10, $|\xi_{min}(I, s) - \xi_{min}(I, s')| \leq \min(|\mathcal{U}(s, s')|, |\mathcal{E}(s, s')|)$ holds.

ii) Let I be a prime node with prime child nodes $I_1, \ldots, I_p$ and $(s_1, \ldots, s_p)$ be a tuple of signs in Equation (3.1) for which $\xi_{\min}(I, s)$ is minimal. Assume that there are k-signs s_i', $i \in [1 : p]$, with $\xi_{\min}(I_i, s_i') < \xi_{\min}(I_i, s_i)$. Let $S \subseteq \{1, .., p\}$ be a non-empty set of indices where $\xi_{\min}(I_i, s_i') < \xi_{\min}(I_i, s_i)$ holds for all $i \in S$ and let $(s_1', \ldots, s_p')$ be the tuple of k-signs generated from $(s_1, \ldots, s_p)$ by replacing s_i by s_i' for all $i \in S$. Let

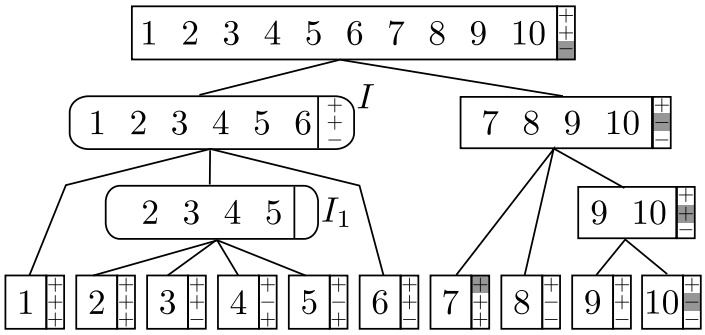


Figure 3.6 – Ambiguous strong interval tree $T^3(\Pi)$ for the permutations of Example 3.3.7; linear nodes are depicted with rectangular boxes, prime nodes are depicted with round boxes; signs indicate the 3-sign of a node, with (s_1, s_2, s_3) from top to bottom; shaded signs indicate inversions that have to be applied because of sign differences between a linear node and its parent

$\mathcal{U}_i = \mathcal{U}(s_i, s'_i)$ and $\mathcal{E}_i = \mathcal{E}(s_i, s'_i)$. From the result of the first part of the proof, it follows that $\sum_{i=1}^{p} \xi_{\min}(I_i, s_i) - \sum_{i=1}^{p} \xi_{\min}(I_i, s'_i) \leq \sum_{i=1}^{p} \min(|\mathcal{U}_i|, |\mathcal{E}_i|)$. By Corollary 3.3.10 it holds

$$|\xi(I, s, s'_1, \ldots, s'_p, s_{p+1}, \ldots, s_{p+l}) - \xi(I, s, s_1, \ldots, s_p, \ldots, s_{p+l})| \leq \sum_{i=1}^{p} \min(|\mathcal{U}_i|, |\mathcal{E}_i|),$$

which means that the number of inversions needed to solve the oIMP in node I is increased maximally by $\sum_{i=1}^{p} \min(|\mathcal{U}_i|, |\mathcal{E}_i|)$ if $(s'_1, \ldots, s'_p)$ is used instead of $(s_1, \ldots, s_p)$.

Hence, by assigning s'_i to node I_i, with $i \in S$, it is in any case possible to find the minimal value for $\xi_{\min}(I, s)$ in Equation (3.1) and consequently sign s_i for node I_i has not to be considered in Equation (3.1).

□

When the minimal score of a pIMP has to be computed, Proposition 3.3.15 can reduce the number of k-sign vectors to be tested in a prime node significantly, especially if a prime node has many prime child nodes. Nevertheless, if all medians have to be computed, all k-sign vectors have to be evaluated in Equation (3.1), as also non-optimal decisions in the prime child nodes may lead to a preserving median scenario.

The case of solving the pIMP for ambiguous trees is illustrated in the following example.

Example 3.3.7. *Let* $\Pi = (\pi_1 \pi_2, \pi_3)$ *be a sequence of three input permutations with* $\pi_1 =$ (1 2 3 4 5 6 7 8 9 10)*,* $\pi_2 =$ (-5 -4 2 3 6 1 9 -10 -8 7)*, and* $\pi_3 =$ (-10 -9 -8 7 4 2 5 -3 1 -6)*. The non-trivial strong common intervals for these gene orders are* $C = \{\{1, 2, 3, 4, 5, 6\},$

$\{2,3,4,5\}$, $\{7,8,9,10\}$, $\{9,10\}\}$. *The* 3*-signed strong interval tree* $T^3(\Pi)$ *is depicted in Figure 3.6.*

The strong interval tree has four linear nodes with parents having a different 3*-sign (*$\{7\}$, $\{7,8,9,10\}$, $\{9,10\}$, *and* $\{10\}$*). Furthermore, the* 3*-sign of the linear root node is not equal to* $(+,+,+)$*. Together this implies five necessary inversions for the linear nodes.*

There are two prime nodes in this tree, namely $I_1 = \{2,3,4,5\}$ *and its parent node* $I = \{1,2,3,4,5,6\}$*. Note that the index* 1 *of node* I_1 *is chosen with respect to the numbering scheme used in Equation* (3.1)*. The* 3*-sign of prime node* I_1 *is unknown since its parent* I *is a prime node. Furthermore, the sign of the element* 2 *which corresponds to the prime child node* I_1 *in the signed quotient permutations of* I *is unknown. This is marked by '*$\pm$*' in the signed quotient permutations. The signed quotient permutations of prime node* I_1 *are* $\pi_{I_1|1}$ = (1 2 3 4), $\pi_{I_1|2}$ = (-4 -3 1 2), *and* $\pi_{I_1|3}$ = (3 1 4 -2) *and for prime node* I *they are* $\pi_{I|1}$ = (1 $\pm$ 2 3), $\pi_{I|2}$ = ($\pm$2 3 1), *and* $\pi_{I|3}$ = ($\pm$2 1 -3)*. For each of the four* 3*-sign assignments* $s_1 \in \Sigma'$ *for* I_1 *the oIMP is solved. The score* $\xi_{\min}(I_1, s_1)$ *and the corresponding medians are given in Table 3.1. Furthermore, Table 3.1 shows the medians and scores of the four* 3*-sign assignments not in* Σ'*, which do not have to be computed according to Proposition 3.3.13. The* 3*-sign for* I_1 *defines the sign of element* 2 *in the quotient permutation of* I*. The solutions of the oIMP of node* I*, when element* 2 *is signed according to the* 3*-sign* s_1 *of* I_1*, are also given in Table 3.1. In this example the* 3*-signs* $(+,+,+)$ *and* $(+,-,-)$ *lead to a minimal overall score* $\xi_{\min}(I, s) = 9$ *for the subtree induced by the prime nodes. The computation of the medians* (1 -6 2 3 4 5 -7 8 9 10) *and* (-5 -4 -3 -2 -1 -6 -7 8 9 10) *by applying the permutations implied by oIMP solutions and inversions resulting from sign differences between linear nodes starting from* π_1 *is shown in Figure 3.7. For those* k*-signs, where* $\xi_{min}(I, s)$ *is minimal, all combinations of oIMP medians of* I_1 *and* I *have to be applied in order to find all medians. But it is possible that different combinations lead to the same median.*

The median for the pIMP was determined by solving four oIMPs for I_1 *and four oIMPs for* I*. The overall score is* 14*.*

If one wants to compute only one median, three of the oIMPs for I *do not have to be computed. The oIMP for prime node* I_1 *has a unique minimum score of four for* 3*-sign* $(+,-,-)$*, the other* 3*-signs yield a score of five. According to Proposition 3.3.15 it is sufficient to consider only this* 3*-sign for the computation of the oIMP for* I*. Hence, for prime node* I *only one oIMP has to be solved. Altogether, only five of the eight oIMPs have to be solved.*

3.3.5 Pseudo code of algorithm TCIP, run time analysis, and extensions

The pseudo code for TCIP with three input permutations is given in Algorithm 4 for the case that the output is a single preserving median of the given input permutations. In the

3-sign s_1 of I_1	$\xi_{\min}(I_1, s_1)$	oIMP medians (I_1)	$\xi(I, s, s_1, s_2, s_3)$	oIMP medians (I)	$\xi_{\min}(I, s)$
$(+,+,+)$	5	(-4 -3 -2 -1)	4	(1 -3 -2)	9
$(+,+,-)$	5	(1 2 3 4)	5	(-3 -2 1), (1 -3 -2)	10
$(+,-,+)$	5	(-2 -1 3 4), (1 2 3 4)	5	(-2 -3 1), (1 2 3), (3 -1 -2), (1 -3 -2)	10
$(+,-,-)$	4	(1 2 3 4)	5	(-2 -1 -3), (1 -3 2)	9
$(-,-,-)$	5	(1 2 3 4)	4	(1 -3 2)	9
$(-,-,+)$	5	(-4 -3 -2 -1)	5	(-3 2 1), (1 -3 2)	10
$(-,+,-)$	5	(-4 -3 1 2), (-4 -3 -2 -1)	5	(2 -3 1), (1 -2 3), (3 -1 2), (1 -3 2)	10
$(-,+,+)$	4	(-4 -3 -2 -1)	5	(2 -1 -3), (1 -3 -2)	9

Table 3.1 – Median computation for prime nodes I and I_1 of Example 3.3.7; given are the 3-sign assignments s_1 of I_1 (first column), the score $\xi_{\min}(I_1, s_1)$ for node I_1 (second column), all medians of the induced oIMP in node I_1 (third column), the score $\xi(I, s, s_1, s_2, s_3)$ when signs for element 2 are chosen according to the 3-sign s_1 (fourth column), all medians of the induced oIMP in node I (fifth column), and score $\xi_{\min}(I, s)$ for prime node I (sixth column); the table shows the results for all eight possible 3-sign assignments, where the top four rows are the 3-sign assignments to be tested, the bottom four rows are only for illustration; note that the medians in the bottom four rows can be directly derived from the medians in the first four rows by inversion of complete permutations (third column) or single elements — element 2 in this example — (fifth column)

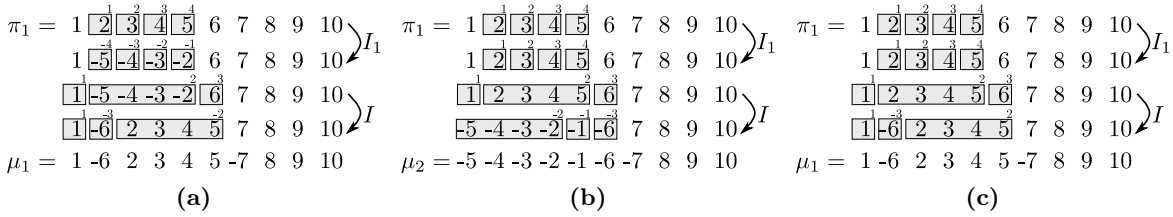


Figure 3.7 – Computation of the two preserving medians for Example 3.3.7; shown is the computation for the combinations of medians of the oIMPs for I and I_1 found with $\xi_{\min}(I, s) = 9$; a) $s_1 = (+, +, +)$, b) $s_1 = (+, -, -)$ using (-2 -1 -3) as median of the oIMP defined for I_1, and c) $s_1 = (+, -, -)$ using (1 -3 2) as median of the oIMP defined for I_1; in each of the three sub figures the following is shown: the first row shows π_1, after this from top to bottom: the permutations induced by the oIMP medians with 3-sign assignment s_1 of prime nodes I_1 and I are applied, and finally one inversion due to the sign differences between the linear node $\{7\}$ and its linear parent $\{7, 8, 9, 10\}$ is applied, resulting in median μ_1 (in a) and c)) and μ_2 (in b)); the boxes represent the child nodes of the prime nodes and the small numbers above the boxes represent the elements of the corresponding quotient permutation of the prime nodes (respectively the prime node median)

freely available implementation of TCIP, the computation of a single preserving median as well as the computation of all preserving medians is possible. For computing all preserving medians the pseudo code has to be modified, such that it iterates over all 3-sign vectors $(s_1, \ldots, s_p)$ for which the property according to line 25 of the pseudo code holds. If only one preserving median has to be computed, it is sufficient to compute only one median in the induced oIMPs. This may also reduce the run time significantly. Note again that TCIP is currently implemented only for $k = 3$. For solving oIMPs the implementation of TCIP uses Caprara's IMP solver, included in the GRAPPA package (Moret et al., 2002a,b, 2001) which is a very efficient median IMP solver.

Before the run time is analysed, note that the pseudo code presented here differs slightly from the pseudo code presented in Bernt et al. (2008b) where the linear nodes have been handled in a first iteration over the strong interval tree and the prime node components in a second one. As a consequence, changes in the k-signs made in the first pass had to be propagated in the subtrees. This adds a linear factor to the run time, which is saved in the given modification.

In the following, the run time of Algorithm TCIP for k input permutations of length n is analysed for the case of unambiguous trees with no prime nodes. The strong interval tree can be built in time $\mathcal{O}(kn)$ for k permutations of size n (Bérard et al., 2007; Bergeron et al., 2008a). Algorithm TCIP has to iterate over all nodes of the strong interval tree. There are $\mathcal{O}(n)$ nodes in a strong interval tree with n leaves. In order to determine the set

Algorithm 4: TCIP(π_1, π_2, π_3)

Input: signed permutations $\Pi = (\pi_1, \pi_2, \pi_3)$
Output: a preserving median μ

Initialise the traces $\pi'_k \leftarrow \pi_k, k \in [1:3]$, let $\Pi' \leftarrow (\pi'_1, \pi'_2, \pi'_3)$;
Compute the 3-signed strong interval tree $T^3(\Pi, \Pi')$;
forall *nodes I in $T^3(\Pi, \Pi')$ (traversal bottom-up)* **do**
  `// Linear node handling`
  if *I linear with linear parent J with ($s(I) \neq s(J)$ and $s(I) \neq -s(J)$)* **then**
    Let ρ be the inversion corresponding to node I that has to be applied to trace π'_k, such that $s(I) = s(J)$ or $s(I) = -s(J)$ holds after inverting the k-th entry of the 3-sign of I (compare Theorem 3.3.5);
    $\pi'_k \leftarrow \pi'_k \circ \rho$, update Π';
  `// Prime node handling`
  else if *I is prime and has no prime parent* **then**
    $T_p \leftarrow$ prime node component with root node I;
    forall *prime nodes I in T_p (traversal bottom-up)* **do**
      Let $I_1, \ldots, I_p$ be the prime children of I;
      Let $I_{p+1}, \ldots, I_{p+l}$ be the linear children of I with 3-signs $s_{p+1}, \ldots, s_{p+l}$;
      forall *3-signs $s \in \Sigma$* **do**
        $\xi_{\min}(I, s) \leftarrow \infty$;
        forall *3-sign vectors $(s_1, \ldots, s_p) \in \Sigma^p$* **do**
          Solve the oIMP for $\Pi_{I|s_1,\ldots,s_p,s_{p+1},\ldots,s_{p+l}}$ with k-sign s;
          Let $\xi(I, s, s_1, \ldots, s_{p+l})$ be the score of this oIMP;
          if $\xi(I, s, s_1, \ldots, s_{p+l}) + \sum_{i=1}^{p} \xi_{\min}(I_i, s_i) < \xi_{\min}(I, s)$ **then**
            $\xi_{\min}(I, s) \leftarrow \xi(I, s, s_1, \ldots, s_{p+l}) + \sum_{i=1}^{p} \xi_{\min}(I_i, s_i)$;
    forall *prime nodes I in T_p (traversal top-down)* **do**
      if *the root node I of T_p has a linear parent* **then**
        assign the 3-sign of the linear parent to node I;
      else `// in this case` I `is the root of` $T^3(\Pi, \Pi')$
        assign the 3-sign $(+, +, +)$ to node I;
      Let s be the 3-sign of node I and $I_1, \ldots, I_p$ be its prime node children;
      Let $(s_1, \ldots, s_p)$ be a 3-sign vector with $\xi_{\min}(I, s) = \xi(I, s, s_1, \ldots, s_{p+l}) + \sum_{i=1}^{p} \xi_{\min}(I_i, s_i)$;
      for *$i = 1$ to p* **do**
        assign the 3-sign s_i to I_i;
    apply all inversions to $\pi'_k, k \in [1:3]$ induced by the oIMP solutions, update Π';
if *root node I has not the 3-sign $(+, +, +)$ or $(-, -, -)$* **then**
  apply an inversion to one trace, such that the signs of the 3-sign of I become identical;
`// all traces` $\tau_k, k \in [1:3]$ `end in a median` μ

of permutations for which a node has to be inverted, the k-sign has to be compared to the k-sign of the parent or to $\{+\}^k$ if the node is the root. That is $\mathcal{O}(k)$ sign comparisons have to be done for each node. Therefore, it is possible to determine the number of inversions of a preserving median scenario, i.e. the score, in time $\mathcal{O}(kn)$. To compute a median scenario and the corresponding traces, the inversions of the preserving median scenario have to be applied for each of the k input permutations. When only the median is of interest, it suffices to apply the inversions on one of the permutations. The length of a preserving median scenario is in $\mathcal{O}(kn)$ because $\lfloor k/2 \rfloor$ inversions have to be applied at most in each of the $\mathcal{O}(n)$ nodes. If the permutations are stored in arrays, it takes time proportional to the length of an inversion in order to perform it, i.e. $\mathcal{O}(n)$. Ergo it would take time $\mathcal{O}(n^2)$ to find a median, given a preserving median scenario. The time needed for the application of an inversion can be reduced to $\mathcal{O}(1)$ by representing the genomes as doubly linked lists, as pointed out in Kaplan and Verbin (2003). More technically, this can be realised as follows (not shown in the pseudo code). Each of the permutations is stored as a double-linked list with the elements as node entries. A traversal of the list gives the order of the elements of the permutation. In order to represent signs of the elements, a boolean flag is stored at each node that indicates if a sign switch happens during the traversal or not. In each node of the strong interval tree, the pointers to the first and last element of the corresponding strong common interval are stored for each of the k permutations. These pointers are determined when the k-signed strong interval tree is computed in linear time. Applying an inversion to a permutation is done by i) inverting the sign switch flag of the first element of the inversion and of the element following the last element of the inversion and ii) relinking the corresponding pointers in the permutations. Hence, an inversion corresponding to a linear node can be applied in constant time.

Theorem 3.3.16. *Let Π be a sequence of k signed permutations of length n. If $T^k(\Pi)$ has no prime node then a preserving median scenario as well as a preserving median can be computed in time $\mathcal{O}(kn)$.*

The run time for determining all preserving medians depends on their number. For an odd number of permutations there is always only one median — see Corollary 3.3.9. For an even number of permutations the number of medians is in $\mathcal{O}(2^a)$, where a is the number of nodes with equally sized sets of equal signs and unequal signs — see Theorem 3.3.8. Computing all preserving median scenarios is problematic also because the inversions of each preserving median scenario do commute, i.e. the number of preserving scenarios is in $\mathcal{O}(n!)$ for each median.

If there exist prime nodes in the strong interval tree, the induced oIMPs have to be solved. Recall that solving the oIMP is NP-hard; with Caprara's median solver (Caprara, 2003)

oIMP instances with input permutations of size up to ≈ 50 can often be solved within seconds nevertheless.

Number of oIMPs to be solved

In the following, the number of oIMPs that have to be solved for a given strong interval tree are analysed.

If a prime node I has p prime child nodes $I_1, \ldots, I_p$. $2^{(k-1)p}$ different k-sign vectors $(s_1, \ldots, s_p)$ have to be considered for these child nodes. For each combination of a k-sign $s \in \Sigma$ for I and a k-sign vector for the prime child nodes, one oIMP has to be solved (compare the loop starting in line 12 of Algorithm 4). An exception is the case when I is the root or I has a linear parent where the k-sign is already known. The total number of oIMPs that have to be solved in the worst case is $\sum_{I \in \mathcal{P}} 2^{(k-1)(p(I)+r(I))}$, where $\mathcal{P}$ is the set of all prime nodes in the strong interval tree, $p(I)$ is the number of prime child nodes of node I, and $r(I) = 1$ if I has a prime node parent and otherwise $r(I) = 0$. It should be mentioned that the number of oIMPs to be solved per prime node can potentially be reduced according to Proposition 3.3.15 if only one preserving median has to be computed (not shown in Algorithm 4). If, for example, each child node I_i has a unique minimum $\xi_{\min}(I_i, s_i)$, only 2^{k-1} oIMPs have to be solved for I. Therefore, only $\sum_{I \in \mathcal{P}} 2^{(k-1)r(I)}$ oIMPs have to be solved in the best case. This is a remarkable improvement compared to the brute force approach because the number of oIMPs that have to be solved is only exponential in the number of prime child nodes, and no longer exponential in the size of the prime node components and the number of prime child nodes.

Recall that the size of the oIMPs induced by the prime nodes is given by the size of its quotient permutations. The number of oIMPs to be solved in the worst case can be calculated in linear time by the formula given above. Thus, the size of the quotient permutations and the number of oIMPs to be solved can be used for an estimation of the run time of TCIP.

Enumeration of all preserving medians

The computation of all preserving medians after solving the oIMPs can lead to a large run time simply because there can exist many medians. There are two factors influencing the number of medians. First, each of the oIMPs may have more than one optimal solution. Second, all possible combinations of the solutions of k-sign assignments to a prime node component with a minimal score can lead to different medians and have to be tested. Theoretically, in each prime node I it may hold that $\xi_{\min}(I, s)$ has the same value for all k-signs $s \in \Sigma$. In this case, each combination of k-sign assignments to prime nodes has to be analysed, as each different k-sign in each prime node may lead to a different preserving median. Hence, the number of k-sign combinations to be analysed is $\Pi_{I \in \mathcal{P}} 2^{(k-1)p(I)}$ in the worst case, where

from each combination a set of preserving medians can be inferred from the solutions of the corresponding oIMPs in the prime nodes. If only one preserving median has to be computed, it is enough to analyse only one k-sign combination per prime node component. In Algorithm 4 (lines 25 to 27) only one k-sign vector $(s_1, \ldots, s_p)$ is assigned to the prime node children.

Since the number and size of prime nodes strongly influence the run time of algorithm TCIP, the occurrence of prime nodes in the strong interval tree is analysed in detail in Section 3.4.

Extensions

Only the case of non circular, i.e. linear, signed gene orders has been considered so far. But for evolutionary scenarios the input permutations might be circular (i.e. all circular shifts of a gene order are assumed to represent the same genome) or linear. Furthermore, the permutations can be directed, (i.e. $\pi \neq -\pi$) or undirected (i.e. $\pi = -\pi$). Until now the linear directed case has been discussed.

For linear undirected input permutations, only the handling of the root node has to be changed: i) linear root nodes can be left unchanged, since the relative orientation of the three permutations does not matter, ii) for prime root nodes all possible k-sign assignments in Σ' have to be tested (instead of only $\{+\}^k$) and the assignments leading to a minimal score have to be chosen.

A solution for the circular undirected case can be derived easily from the linear directed case as follows: Let $(\pi_1, \ldots, \pi_k)$ be k circular undirected gene orders with a preserving median scenario S. By inverting the complete permutation and applying circular shift, there exist k gene orders $\varpi_i = \pi_i, i \in [1 : k]$, such that the first element of ϖ_i is 1. Note that in a circular scenario each inversion can be replaced by its circular complement without changing the score or the resulting median. Hence, for $\varpi_i = \pi_i$ exists a preserving scenario S' with the same score as S and such that no inversion $\rho(i, j)$ has a start index i that is larger than its end index j. S' is also a preserving median scenario under the assumption that ϖ_i are linear and directed. Therefore, the circular undirected case can be solved by shifting and inverting the input permutations as explained. Note that it is not necessary to modify the handling of the root node because the strong interval tree for the shifted and inverted permutations has a linear root node with k-sign $\{+\}^k$.

Using the PC tree data structure (Hsu and McConnell, 2003; Shih and Hsu, 1999) which is a generalisation of the PQ tree data structure, would be an alternative for the handling of circular permutations. So far, the use of the PC tree data structure has not been explored, because PQ trees allow to solve the pIMP for the circular case in an exact and efficient way. The circular directed has not yet been discussed.

3.4 Results

In this section it is empirically shown that many common intervals exist and that IMP solutions are often not preserving. Furthermore, an empirical run time analysis of algorithms CIP, ECIP, and TCIP is performed and the improvements of ECIP and TCIP that have been introduced on the basis of the theoretical results are evaluated. Properties of the strong interval trees for artificial and biological data are investigated. As test instances mitochondrial gene order data from *Metazoan* species of various taxa, chloroplast gene orders from *Campanulaceae*, and simulated gene orders have been used.

3.4.1 Data sets and experimental setup

Construction of random data sets

The random test instances were generated as follows. Starting with the identity permutation of length n, r random inversions were applied to generate a random permutation. Three permutations generate a pIMP instance. $R = \{(n, r) : n \in \{20, 40, \ldots, 100\}, r \in \{1, 2, \ldots, 10\} \cup \{n/20, 2n/20, \ldots, 6n/20\}\}$ is the set of all considered combinations of n and r. Set R contains 62 different combinations of n and r. For each $(n, r) \in R$, 1 000 pIMP instances have been generated. This results in 62 000 random data sets.

Biological data sets

As biological data the improved mitochondrial gene orders given in Appendix A are used. These gene orders have usually the standard set of 37 mitochondrial genes consisting of 13 protein coding-, two rRNA-, and 22 tRNA- genes. An exception are the gene orders from *Nematodes* and *Platyhelminthes* where nearly all have no gene ATP8, hence a reduced set of 36 genes is used for these groups. The considered phylogenetic groups are as follows:

i) From the superphylum *Deuterostomia*: the *Chordata* (cho), the and the remaining *Deuterostomia*, i.e. *Echinodermata*, *Hemichordata*, *Xenoturbellida* (ehx) are used. Furthermore, the *Chordates* from the class *Actinopterygii* (act) and the remaining *Chordates* (nac) are considered.

ii) From the *Protostomia*: the *Arthropoda* (art), the combined groups of *Annelida*, *Brachiopoda*, *Echiura*, *Mollusca*, and *Sipuncula* (abm), and the *Nematodes* and the *Platyhelminthes* (np) form a group. From the *Arthropoda* the subgroups *Crustacea* (cru), *Hexapoda* (hex), and the combined *Chelicerata* and *Myriapoda* (cmy) are used.

Additionally, the chloroplast gene orders (cp) from Cosner et al. (2000a) are used (the data set is included in the GRAPPA distribution (Moret et al., 2002a,b, 2001)). This data set consists of 13 distinct gene orders, each having 105 genes.

For each taxonomic group all possible subsets of size three of unique gene orders were used as test instances. The number of different gene orders within each group can be found in Table 3.4 (#). Altogether, there are 132 432 test instances for the mitochondrial gene orders. Note, this data set differs from the data set used in Bernt et al. (2008b). First, it is up to date, i.e. some of the considered groups contain now twice as much unique gene orders and the number of data sets is nearly four (3.65) times more. Second, the gene orders included in both data sets may differ because of improvements of the GenBank data base and the additional improvements for tRNA locations as described in Appendix A. For the chloroplasts 286 data sets are considered.

Experimental setup

All test runs were done on PCs with an AMD Opteron 2.0 GHz processor and 4 GB of main memory. A run time limit of one hour was applied for each data set.

3.4.2 Destroyed common intervals

The number of common intervals for the test instances in the different taxa is shown in Figure 3.8. The figure shows that the number of common intervals in the test sets is very different between the groups of the biological data sets. For data sets in the group containing the *Mollusca* the average number of common intervals is 36.3 and it is nearly 600 for the Chordata. The number of common intervals is higher for *Deuterostomia* (cho, act, nac, ehx) as for *Protostomia* (cru, hex, cmy, art, abm, np). This indicates that the available gene orders of *Deuterostomia* are less diverse than the available gene orders from *Protostomia*. For the simulated data sets it can be seen that for data sets generated with a small number of inversions r have a larger number of common intervals. The number of common intervals decreases fast with increasing values of r.

Figure 3.8 depicts the fraction of the common intervals that are destroyed by an optimal solution of the IMP for those test instances where at least one common interval is destroyed. For computing the fraction the maximum number of destroyed common intervals was used for each instance. Obviously for most taxa the fraction of destroyed common intervals is larger than 0.2. An exception are the *Arthropoda* groups (except the *Hexapoda*) where the fraction is smaller. Moreover, in most taxa there exist test instances where approximately 50% of the common intervals are destroyed. Interestingly, for the groups which have a smaller number of common intervals, e.g. abm and np, a large fraction of destroyed common intervals is observed, too. This can also be observed for the simulated data sets, where for data sets generated with a few inversions, i.e. large number of common intervals, no intervals are destroyed. Whereas for the data sets generated with more inversions, i.e. less common intervals, the fraction of destroyed common intervals increases. The percentage of test instances,

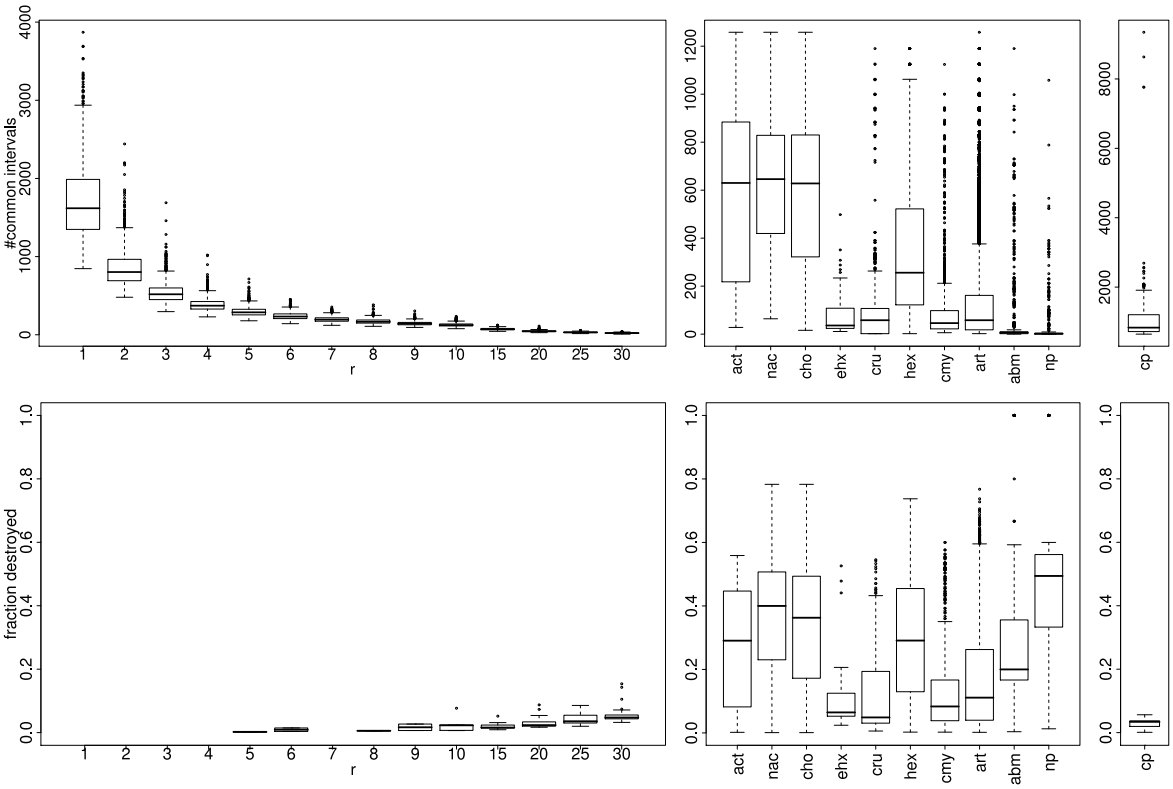


Figure 3.8 – Top: number of common intervals for the data sets; bottom: fraction of common intervals that are destroyed by an exact solution of the IMP, data sets where no median destroys a common interval are excluded otherwise for each data set the maximum number of destroyed intervals is used; from left to right: the simulated data sets with $n = 100$, mitochondrial gene orders, chloroplast gene orders

where at least one common interval is destroyed by an optimal solution of the IMP, is shown in Table 3.2. For the simulated data sets the percentage is very small, but increases with larger values of r. Apparently in most taxa of the biological data sets the percentage is nearly 20% or more. Only for *Mollusca* and *Echinodermata* the percentage is less than 20%. It can be concluded that for nearly all investigated taxa there exist a significant fraction of test instances where many of the existing common intervals are destroyed by an optimal solution of IMP. This shows the relevance of the pIMP.

3.4.3 Computing pIMP medians

In this section the run times of the presented pIMP solvers, i.e. CIP, ECIP, and TCIP are compared. For the direct comparison only 100 randomly chosen data sets are used for each combination of n and k in the simulated data sets and for each group of the biological data sets. The reason for this is the large run time of ECIP and especially CIP as discussed below.

r	%
1	0.0
2	0.0
3	0.0
4	0.0
5	0.1
6	0.2
7	0.0
8	0.1
9	0.2
10	0.5
15	1.5
20	3.0
25	5.2
30	3.7

group	%
act	30.3
nac	46.6
cho	42.1
ehx	12.1
cru	22.8
hex	49.4
cmy	28.0
art	32.2
abm	19.0
np	25.6
cp	30.8

Table 3.2 – Percentage of test instances where at least one common interval is destroyed by an optimal solution of the IMP; left: for the biological data sets; right: for the simulated data sets with $n = 100$

Boxplots showing the run times of the three pIMP solvers are given in Figure 3.9. In order to have a relation for the run time of the pIMP solvers, the boxplots also include the run time of Caprara's median solver — solving the IMP — for the 100 randomly chosen data sets. A comparison of TCIP and Caprara's median solver for the complete data set is presented in the next section.

CIP and ECIP were able to solve most of the data sets within the time limit of one hour. This shows the possibility to solve the pIMP with these methods in reasonable run time. But for some problem instances in the *Arthropoda* and *Crustacea* data set and some simulated data sets with $n \geq 80$ the time limit was exceeded. For example, CIP could not solve 6 out of the 100 data sets from the *Arthropoda* within the time limit. Running CIP and ECIP for all data sets was impossible with the available computing resources. For example, the assumption that 6% of the 73 150 *Arthropoda* data sets have a run time of more than one hour gives a run time estimation of 8.35 years. This is the reason for limiting the number of data sets to 100 per group for the biological data sets and per combination of (n, r) for the simulated data sets. The run times of CIP and ECIP are very similar for the simulated data sets, but one can argue that CIP has slightly better run times. For the majority (4 331) of the simulated data sets CIP is faster than ECIP whereas only for 1 863 data sets ECIP is faster. On average CIP is 1.11 times faster than ECIP for the simulated data sets. But note that while on the one hand CIP is 1.24 times faster than ECIP on average for the 4 331 data sets where the CIP is faster, ECIP is on average 3.44 times faster than CIP for the 1 863 data

sets where ECIP is faster. This is reflected in the sum of the run times. ECIP needs 14% less time than CIP to solve all data sets.

For the biological data sets the run time difference is more pronounced and contrary to the simulated data sets it can be argued that ECIP is faster. This is clearly reflected in the boxplots of the run times, especially for the three *Chordata* groups and the *Hexapoda*. For example ECIP is 7.83 times faster than CIP on average and for 62.5% of the data sets the absolute run time of ECIP is smaller. Note that these are exactly the groups which have the most common intervals (see Section 3.4.2). This is as expected because the more common intervals are to be preserved the more the branch and bound of the search tree of Caprara's median solver can be pruned.

TCIP has the smallest run time for nearly all data sets. Only for 19 of the simulated data sets CIP was faster than TCIP and for 14 of these also ECIP was faster. The difference of the measured run times is less than 7.4 milliseconds except for one of the 19 instances where the run times of CIP and TCIP differed by 1.11 seconds. This maximum difference was for $n = 100$ and $r = 30$. For the biological data CIP was faster than TCIP for 29 data sets. The great majority (i.e. 22) of these differences is observed for data sets of *Nematode*, *Platyhelminthes* and the *Mollusca* group. Four times the run time of CIP is smaller than the run time of TCIP by more than ten seconds with a maximum of 350.32 seconds. The data sets with a smaller run time of CIP (compared to TCIP) also had a smaller run time of ECIP (compared to TCIP). TCIP is the fastest of the pIMP solvers for the remaining data sets, i.e. the vast majority. For the biological data sets TCIP is on average 34 842 (1 078) times faster than CIP (ECIP). For 18% (9.5%) percent of the biological data sets TCIP was 1 000 times faster than CIP (ECIP). Also for the simulated data sets TCIP is faster than CIP and ECIP, on average 221 times in the comparison with ECIP and 1 100 times in the comparison with CIP. For six biological and one simulated data sets TCIP needed more than 10^6 times less run time than CIP to compute a pIMP median. The maximum is $7.6 \cdot 10^6$ measured for a data set in the *Hexapoda* group.

The run time of all median solvers depend on the number of applied inversions, i.e. the diversity of the data set. See for example the run times of the pIMP solvers for each r separately, as shown for $n = 100$ in Figure 3.10. For small values of r the median run time of ECIP is slightly smaller than the median run time of CIP. Recall, for small r the simulated data sets have the most common intervals. For larger values of r the CIP algorithm is slightly faster.

The results show that CIP and ECIP can solve the pIMP for simulated as well as biological data sets in reasonable time. But TCIP runs faster than both by orders of magnitude.

The correctness of the implementation was empirically verified by comparing the results of the three pIMP solvers.

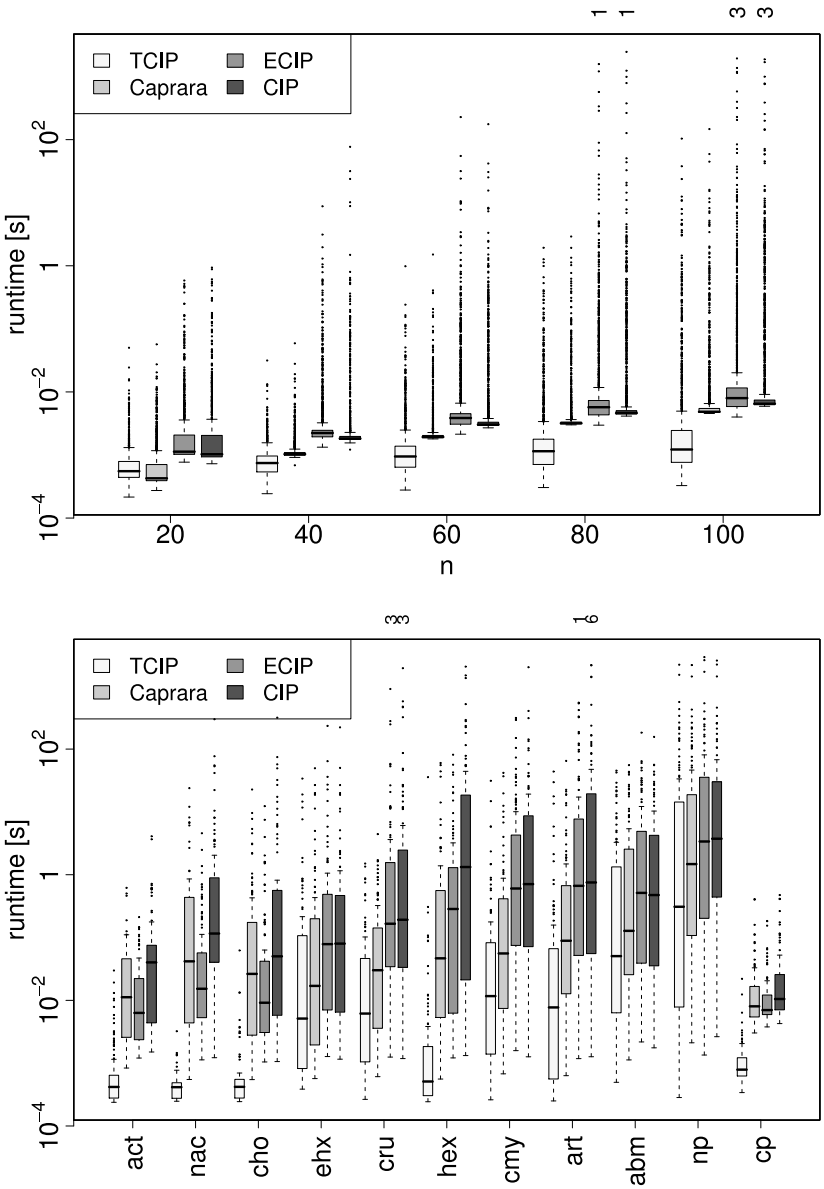


Figure 3.9 – Comparison of the computation times of Caprara's median solver, TCIP, ECIP, and CIP (times are measured in seconds); top: boxplot for the random data sets with $(n, r) \in R$, for 100 randomly chosen test instances per combination of permutation size n and number of inversions r; bottom: boxplots for the biological data sets where 100 data sets have been randomly chosen for each group; small number on the top of the plots give the number of data sets not been solved within the time limit of one hour

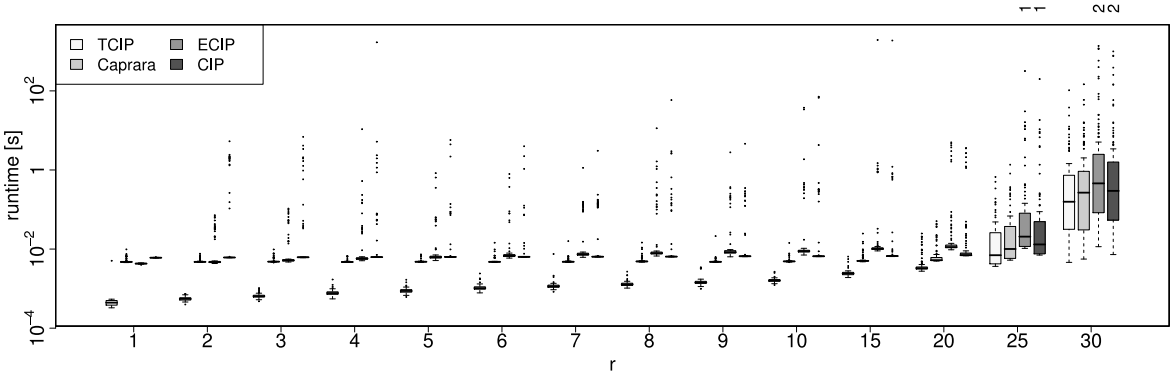


Figure 3.10 – Comparison of the computation times of Caprara's median solver, TCIP, ECIP, and CIP (times are measured in seconds); boxplots for the random data sets with $n = 100$ and $r \in \{1, .., 10, 15, 20, 25, 30\}$, where 100 test instances for each number of inversions r were used; small number on the top of the plots give the number of data sets not been solved within the time limit of one hour

Specific aspects of CIP and ECIP

In this subsection some statistics of characteristic values of Caprara's median solver and the modified algorithms CIP and ECIP are discussed which influence the run time (see Table 3.3). Recall the mode of operation of Caprara's median solver. That is the use of the lower bound of the problem instance as target value t for the branch and bound algorithm which t is increased by one until a solution is found. This procedure was introduced in Caprara's median solver motivated by the observation that the lower bound often coincides with the inversion score of the problem. The inversion distance is a lower bound for the preserving inversion distance. Hence, for a set of input permutations the lower bound as well as the score of a pIMP are at least as big as the lower bound and the score for the IMP. Thus, the number of necessary incrementations of t may be increased or decreased depending on which value grows faster, i.e. lower bound or score. For nearly all considered biological and simulated data sets the value of t has to be increased at least as often as if the IMP is to be solved. The only exception are 16 of biological problem instances where the number of increments is reduced by one. For the biological data sets the average number of incrementations is larger by 0.48 and by 0.25 for the simulated data sets when solved with CIP or ECIP. Furthermore, the results show that while the average number of necessary increments of t is smaller than 1 for the simulated data, it is much larger, i.e. between two and three, for the biological data. That is often the lower bound does not coincides with the score of the problem and indicates that a reevaluation of the target value approach of Caprara's median solver may be beneficial. Another characteristic of the branch and bound median solver influencing the run time is the number of complete permutations which are constructed, i.e. how many leaves of the branch and bound search tree are checked. Obviously the number of checked permutations is drastically increased for

	simulated			biological		
	$avg(\Delta t)$	comp $[\cdot 10^3]$	rec $[\cdot 10^6]$	$avg(\Delta t)$	comp $[\cdot 10^3]$	rec $[\cdot 10^6]$
Caprara	0.159	12	1 595	2.279	3 154	30 101
CIP	0.407	4 000	65 997	2.764	12 160	151 496
ECIP	0.407	4 000	23 685	2.764	12 160	49 512

Table 3.3 – Statistics for the branch and bound algorithm of Caprara's median solver solving the IMP and the modified algorithms CIP and ECIP solving the pIMP; $avg(\Delta t)$: average number of increments of the initial lower bound; comp: number of complete permutations constructed; rec: number of recursion calls; only results for data sets that have been solved by all three median solvers are included

the pIMP. Note, for each constructed permutation the score is computed. For the IMP this involves the computation of three inversion distances which can be computed in linear time whereas for the pIMP three preserving inversion distances, i.e. three NP-hard problems, have to be solved. Each applied static or dynamic constraint saves the exploration of a part of the search tree, i.e. at least one recursion call less is necessary. The static constraints have been applied 2 263 (2 028) million times for the biological (respectively simulated data sets) and the dynamic constraints have been applied 4 208 (7 043) million times. This is reflected by the reduction of the number of recursion calls in ECIP when compared to CIP.

3.4.4 Properties of strong interval trees

In this subsection some properties of strong interval trees that are crucial for the run time of the pIMP solver TCIP are analysed. In Table 3.4 (respectively Tables 3.5 and 3.6) results are presented for the biological (respectively simulated) data sets. For the biological data sets many instances have only linear nodes. For each of the three *Chordate* groups (act, nac, cho) a large fraction of the strong interval trees has no prime node (28-43%). Also for the *Hexapoda* many of strong interval trees have only linear nodes (24%). For random data sets only data sets generated with a few inversions have no prime nodes. The strong interval trees having prime nodes, mostly only have a very small number of them. The maximum number of prime nodes over all test instances is five. The average number of prime nodes over all strong interval trees having at least one prime node in each group of instances (biological taxonomic group or random instances with the same value of r) is at most 1.28. When more than one prime node occurs in a strong interval tree, these prime nodes usually do not occur within one prime node component. Hence, the number of prime nodes with prime node children is small. Therefore, the number of oIMPs to be solved per instance is also small. The worst case that occurred in the data sets, in terms of the number of oIMPs that have to be solved,

	#	p	q	u	l	d
act	28	57.02	1.02 (2)	1.08 (8)	9.68 (24)	0.01 (1)
nac	39	71.87	1.28 (3)	1.29 (9)	5.51 (20)	0.00 (1)
cho	61	69.24	1.16 (4)	1.19 (9)	7.32 (29)	0.00 (1)
ehx	11	100.00	1.16 (3)	1.52 (9)	22.39 (31)	0.06 (1)
cru	20	96.58	1.10 (3)	1.41 (9)	22.18 (35)	0.05 (1)
hex	30	76.06	1.10 (3)	1.12 (8)	13.88 (35)	0.00 (1)
cmy	27	98.56	1.11 (4)	1.19 (9)	22.39 (34)	0.01 (1)
art	77	95.17	1.12 (5)	1.25 (24)	20.98 (35)	0.02 (2)
abm	23	99.21	1.01 (3)	1.01 (3)	31.35 (36)	0.00 (0)
np	13	99.63	1.02 (3)	1.02 (3)	31.03 (35)	0.00 (0)
cp	13	95.45	1.03 (2)	1.03 (2)	15.86 (26)	0.00 (0)

Table 3.4 – Properties of strong common interval trees for the biological data sets; #: number of unique gene orders; p: percentage of data sets for which the corresponding strong interval tree has at least one prime node; q: average number of prime nodes (average over instances with at least one prime node); u: number of oIMPs to be solved; l: size of quotient permutation in the prime nodes; d: number of prime node neighbours that a prime node has in the strong interval tree; for columns $q, \ldots, d$ only instances are taken into account, for which at least one prime node exists; if two values are given in a column, the first value is the average over all instances and the second value (in parentheses) is the maximal value

is a strong interval tree with one prime node component of size four. This case occurred in the simulated data sets with $n = 20$ and $r = 6$, for the data set consisting of:

- π_1 = (15 -18 -17 -16 -13 1 2 3 -12 -11 -10 -9 7 8 -6 -5 -4 -14 19 20),
- π_2 = (-4 -3 -2 -1 -8 -10 -9 7 11 -6 -5 12 13 14 15-16 17 18 19 20), and
- π_3 = (17 13 14 15 1 2 3 4 5 6 7 8 9 10 -12 -11 -16 18 -19 20).

The strong interval tree is shown in Figure 3.11. The four prime nodes of the strong interval tree are organised linearly in a single component. The root prime node of the component is not the root of the strong interval tree. For the root and the bottom prime node four oIMPs and for each of the two inner prime nodes 16 oIMPs have to be solved. Hence, 40 oIMPs have to be solved in order to compute the medians for the pIMP. The maximum number of oIMPs to be solved for a biological data set is 24. This case occurred six times in the art data set and five times in the simulated data sets. In all cases there are three prime nodes that are either included in a linearly organised component or there is one prime node with two prime child nodes. In both cases the root node of the connected prime node component is not the root of the strong interval tree. For all test instances, where the strong interval tree has the maximum of five prime nodes, the number of oIMPs to be solved was five. This is because in the two cases, the prime nodes occur separated. For sets of gene orders that have

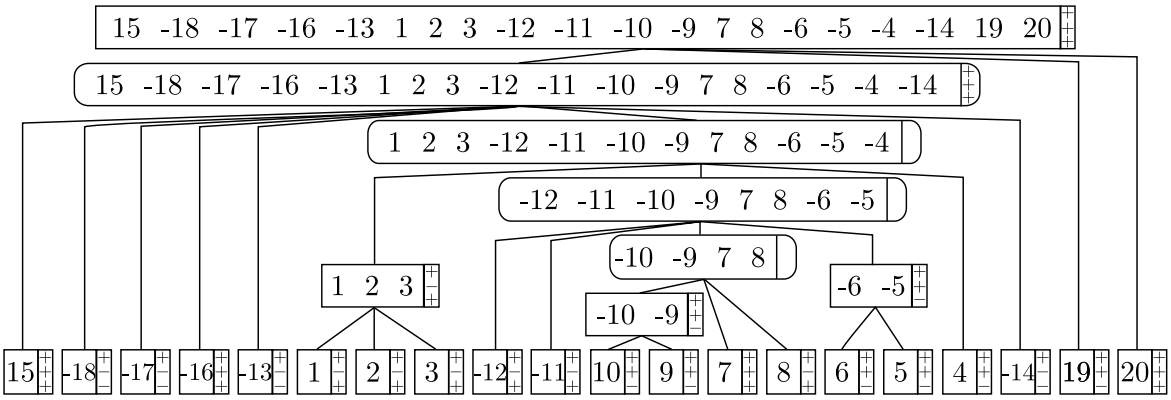


Figure 3.11 – The 3-signed strong interval tree for the pIMP instance found in the data sets with a maximum number of oIMPs to be solved

been simulated with a relatively small number of inversions and the data sets that are similar (measured for example with the number of common intervals) the size of the oIMPs, i.e. the size of the signed quotient permutations, that had to be solved is small compared to the size of the original IMP. For the three *Chordata* groups the average size of the quotient permutations of the prime nodes is less than 10 and for the data sets from the *Hexapoda* it is 13.88. Thus, the average size of the permutations of the oIMPs to be solved is about one third of the size of the original input permutations. The same holds for the sets of randomly generated gene orders with small value of r. For the random instances with $r \leq 10$, the maximal size of a quotient permutation in a prime node is 51. The average size of the quotient permutations of prime nodes increases with the number of inversions that have been applied during the simulation.

Altogether, the results show that many pIMP instances are very suitable for TCIP, i.e. TCIP has to solve only few and small oIMP instances. This is represented by the run times presented in the following section.

3.4.5 Comparison of pIMP and IMP median solvers and medians

In the last section it was shown empirically that TCIP is the fastest of the three algorithms for the preserving inversion median problem presented in this work. In the following the run times of TCIP and Caprara's median solver are compared. Since Caprara's median solver, as presented in Caprara (2003), computes only one median, a slightly modified version of Caprara's median solver is used for the computation of all medians as discussed in Section 2.5.3 and Bernt et al. (2007b). Note, Caprara's median solver does not solve the pIMP but the IMP. The comparison is mainly for assessing how efficient TCIP is compared to another median solver which is known to be fast. The median sets are empirically analysed

r	p	q	u	l	d
1	63.30	1.00 (1)	1.00 (1)	3.69 (5)	0.00 (0)
2	96.50	1.11 (3)	1.21 (8)	7.43 (11)	0.02 (1)
3	99.90	1.11 (3)	1.22 (8)	12.52 (17)	0.02 (1)
4	100.00	1.06 (3)	1.15 (24)	17.67 (23)	0.01 (2)
5	100.00	1.04 (3)	1.06 (8)	22.53 (29)	0.00 (1)
6	100.00	1.04 (2)	1.07 (8)	27.09 (34)	0.01 (1)
7	100.00	1.02 (2)	1.05 (8)	31.39 (39)	0.00 (1)
8	100.00	1.02 (3)	1.05 (8)	35.19 (44)	0.01 (1)
9	100.00	1.02 (2)	1.05 (8)	38.87 (49)	0.01 (1)
10	100.00	1.02 (2)	1.04 (8)	42.42 (51)	0.00 (1)
15	100.00	1.01 (2)	1.04 (8)	56.90 (67)	0.01 (1)
20	100.00	1.00 (2)	1.01 (8)	67.85 (78)	0.00 (1)
25	100.00	1.01 (2)	1.03 (8)	75.54 (86)	0.00 (1)
30	100.00	1.00 (2)	1.03 (8)	81.60 (92)	0.01 (1)

Table 3.5 – Properties of strong interval trees for random data sets; notation see Table 3.4; r: number of inversions applied for generating the data set (1 000 instances for each r); size of the permutations: $n = 100$

in Section 3.4.5 and a comparison of the pIMP and IMP median sets is presented. Also the potential of TCIP as a heuristic for the IMP is discussed there.

In Figure 3.12 boxplots are given for the run times of TCIP and Caprara's IMP solver when one (preserving) median and all (preserving) medians are computed. Apparently, it is faster in most cases to compute one or all preserving inversion medians with TCIP than to compute one or all inversion medians with Caprara's IMP solver.

For a first overview on the run time comparison the sum of the run times is analysed. For this analysis a run time of one hour is assumed for the data sets where the run time limit has been exceeded. Caprara's median solver needed about 6.2 days (534 495 seconds) to compute one median, respectively about 7.1 weeks (4 322 382 seconds) to compute all medians for all biological data sets whereas TCIP needed less than two days (169 364 seconds) for computing one median, respectively about 3.7 weeks (2 265 778 seconds) for computing all medians for the same data sets. For the simulated data sets the differences are less pronounced, but still large, i.e. Caprara's median solver needed about 83.2 minutes (18.4 hours) to compute one median (all medians) and TCIP needed about 54.7 minutes (12.25 hours) to compute one preserving median (all preserving medians).

For a more detailed analysis of the run times the speedup of TCIP and Caprara's median solver is analysed. Speedup is defined here as the fraction of the run time of Caprara's median solver and the run time of TCIP, i.e. the speedup measures how many times TCIP is faster than Caprara's median solver. This notion must not be confused with the definition

r	p	q	u	l	d
1	55.10	1.00 (1)	1.00 (1)	3.55 (5)	0.00 (0)
2	93.80	1.08 (2)	1.23 (8)	6.17 (11)	0.02 (1)
3	99.60	1.11 (3)	1.32 (9)	8.93 (14)	0.03 (1)
4	99.90	1.07 (3)	1.30 (9)	11.28 (18)	0.04 (1)
5	100.00	1.03 (2)	1.15 (8)	13.37 (18)	0.02 (1)
6	100.00	1.04 (4)	1.25 (40)	14.50 (19)	0.03 (2)
7	100.00	1.03 (2)	1.15 (8)	15.57 (19)	0.02 (1)
8	100.00	1.01 (2)	1.04 (8)	16.44 (19)	0.01 (1)
9	100.00	1.01 (2)	1.08 (8)	16.87 (19)	0.01 (1)
10	100.00	1.01 (2)	1.08 (8)	17.37 (19)	0.01 (1)

r	p	q	u	l	d
1	65.60	1.00 (1)	1.00 (1)	3.63 (5)	0.00 (0)
2	96.60	1.11 (3)	1.19 (8)	7.13 (11)	0.01 (1)
3	99.90	1.11 (3)	1.22 (24)	11.73 (17)	0.02 (2)
4	100.00	1.07 (3)	1.21 (24)	16.37 (23)	0.02 (2)
5	100.00	1.04 (2)	1.08 (8)	20.50 (28)	0.01 (1)
6	100.00	1.02 (2)	1.08 (8)	24.39 (33)	0.01 (1)
7	100.00	1.02 (2)	1.05 (8)	27.37 (36)	0.00 (1)
8	100.00	1.02 (2)	1.09 (8)	30.43 (38)	0.01 (1)
9	100.00	1.01 (2)	1.04 (8)	33.12 (42)	0.01 (1)
10	100.00	1.01 (2)	1.04 (8)	35.60 (45)	0.01 (1)
12	100.00	1.01 (2)	1.07 (8)	39.74 (49)	0.01 (1)
15	100.00	1.01 (2)	1.04 (8)	44.73 (53)	0.01 (1)

r	p	q	u	l	d
1	62.30	1.00 (1)	1.00 (1)	3.70 (5)	0.00 (0)
2	96.10	1.09 (2)	1.18 (8)	6.88 (11)	0.01 (1)
3	99.80	1.10 (3)	1.23 (9)	11.02 (17)	0.02 (1)
4	100.00	1.07 (4)	1.21 (10)	14.93 (21)	0.02 (1)
5	100.00	1.05 (3)	1.21 (24)	18.33 (25)	0.02 (2)
6	100.00	1.03 (3)	1.12 (9)	21.25 (29)	0.02 (1)
7	100.00	1.02 (2)	1.08 (8)	23.71 (31)	0.01 (1)
8	100.00	1.02 (2)	1.10 (8)	25.67 (32)	0.01 (1)
9	100.00	1.01 (2)	1.05 (8)	27.72 (34)	0.01 (1)
10	100.00	1.01 (2)	1.06 (8)	29.07 (36)	0.01 (1)
12	100.00	1.00 (2)	1.03 (8)	31.64 (38)	0.01 (1)

r	p	q	u	l	d
1	62.60	1.00 (1)	1.00 (1)	3.71 (5)	0.00 (0)
2	96.60	1.10 (3)	1.17 (8)	7.46 (11)	0.01 (1)
3	99.90	1.11 (4)	1.22 (10)	12.40 (17)	0.02 (1)
4	99.90	1.08 (3)	1.19 (9)	17.07 (23)	0.02 (1)
5	100.00	1.04 (3)	1.09 (8)	21.73 (29)	0.01 (1)
6	100.00	1.04 (2)	1.08 (8)	25.75 (33)	0.01 (1)
7	100.00	1.03 (3)	1.07 (9)	29.82 (38)	0.01 (1)
8	100.00	1.02 (2)	1.06 (8)	33.42 (41)	0.01 (1)
9	100.00	1.01 (2)	1.05 (8)	36.81 (45)	0.01 (1)
10	100.00	1.01 (2)	1.04 (8)	39.67 (49)	0.01 (1)
12	100.00	1.01 (2)	1.02 (8)	45.09 (55)	0.00 (1)
16	100.00	1.00 (2)	1.01 (8)	53.81 (64)	0.00 (1)
20	100.00	1.00 (2)	1.02 (8)	60.33 (72)	0.00 (1)
24	100.00	1.00 (2)	1.01 (8)	65.02 (75)	0.00 (1)

Table 3.6 – Properties of strong interval trees for random data sets with permutations of size $n = 20$ (top left), $n = 40$ (top right), $n = 60$ (bottom left), $n = 80$ (bottom right); notation see Table 3.5

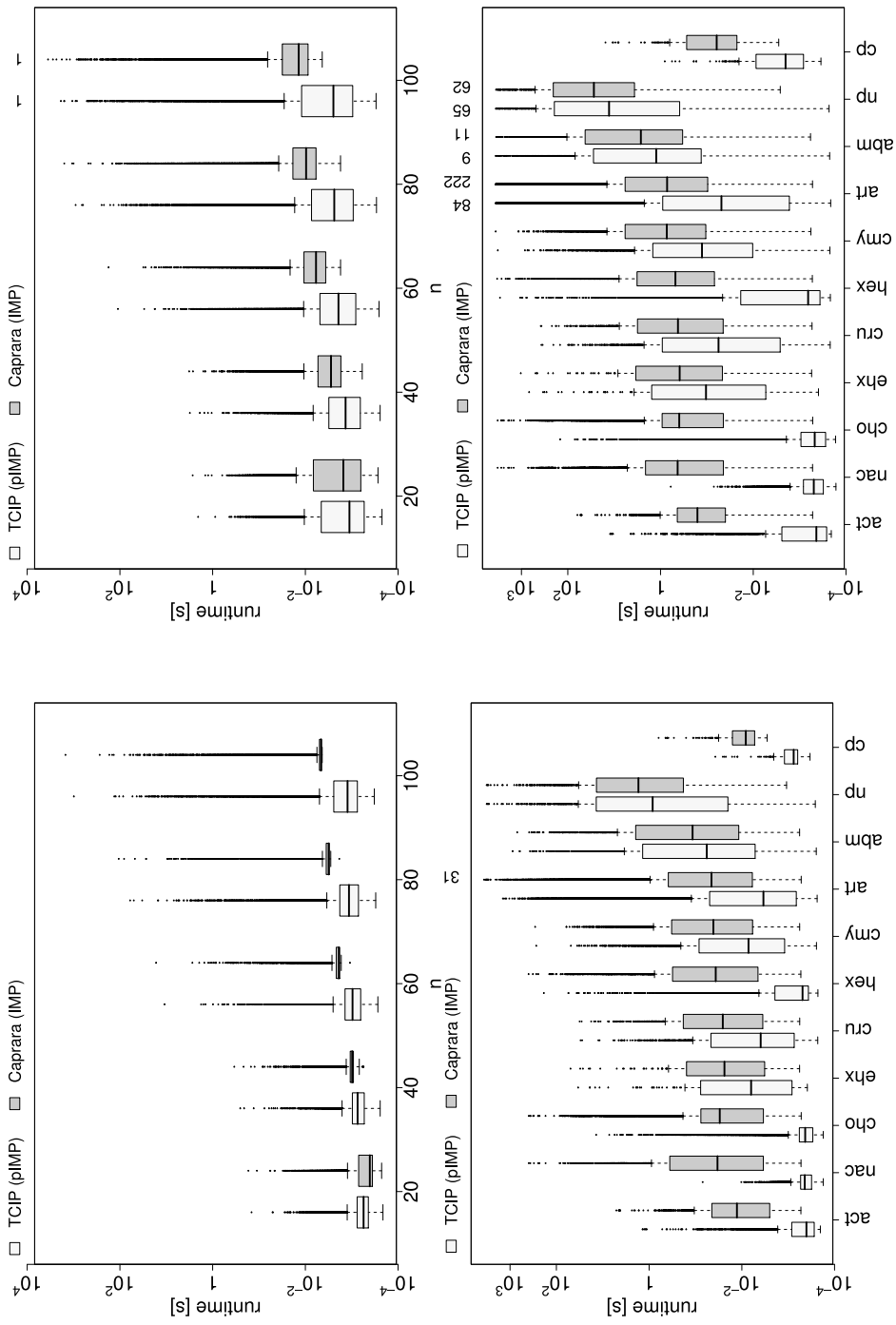


Figure 3.12 – Comparison of the computation times of TCIP solving the pIMP and Caprara's median solver for the IMP (times are measured in seconds); top: boxplot for the random data sets with $(n, r) \in R$, where all 1 000 test instances for each combination of permutation size n and number of inversions r were used; bottom: boxplots for the biological data sets; depicted are computation times for computing one (preserving) median (left) and all (preserving) medians (right); small number on the top of the plots give the number of data sets not been solved within the time limit of one hour

of speedup in parallel computing. Data sets that have not been solved by one of the two methods are excluded in the following analysis. Average, median value, and maximum values of the speedup of TCIP and Caprara's median solver for computing one and all (preserving) medians are presented in Table 3.7. The best results of TCIP with respect to the run time have been found for the data sets in the three considered *Chordata* groups and the *Hexapoda* group. For these groups the median value of the speedup values is larger than 9.8 and for the *nac* group even 74.8 when one (preserving) median is to be computed. In case all (preserving) medians have to be computed the median speedup value of TCIP relative to Caprara's median solver is between 8 to 10.9 times larger than the median speedup value observed for the computation of one (preserving) median. The average values of the speedup are very high for the four groups, i.e. up to 2 437 for computing one (preserving) median and up to 13 917 for computing all (preserving) medians. Note that the average values of the speedup are strongly affected by the extremely large maximal speedup values of nearly 1.4 million for the one (preserving) median case and more than 10 million for the all (preserving) medians case. For the three *Chordata* groups TCIP was mostly faster than Caprara's median solver (see columns g and g_{max} in Table 3.7), i.e. for more than 99.54% of the data sets. Furthermore, in the cases where Caprara's median solver is faster the run time difference is very small, i.e. smaller than three hundredth seconds. About five percent of the *Hexapoda* data sets are computed faster by Caprara's median solver. But only for one data set a large run time difference was measured, i.e. 144.32 seconds, for the remaining instances Caprara's median solver was faster by not more than eight seconds. In the all (preserving) medians case for two data sets a run time difference of nearly or more than 100 seconds was observed (92 and 2 043 seconds), for the remaining data sets Caprara's median solver was faster than TCIP by less than 45 seconds. Thus, TCIP is faster than Caprara's median solver by orders of magnitude for the three *Chordata* data sets and the *Hexapod* data set.

For the remaining data sets TCIP is also faster than Caprara's median solver, but the run time differences are smaller. For the remaining *Arthropoda* data sets (cru, cmy, art) and the *Echinodermata* data set (ehx) TCIP shows a good run time behaviour, i.e. a speedup of at least 1.7 to 3.2 for 50% of the data sets and less than 14% of the data sets are solved faster with Caprara's median solver. For the data set including the *Mollusca* and the *Nematodes* and *Platyhelminthes* data sets, TCIP performed worst, but still comparable run times as Caprara's median solver. This is for both groups more than half of the data sets have been solved faster with TCIP (see columns s_{med} and g in Table 3.7). But for up to 30% of the data sets Caprara's median solver had a smaller run time, where for a considerable fraction of those data sets the difference is not negligible. For example, Caprara's median solver was faster than TCIP by at least 10 seconds for 21 of the np data sets when one (preserving) median is to be computed. However, for 20.6% (respectively 8.7%) of all instances of the biological

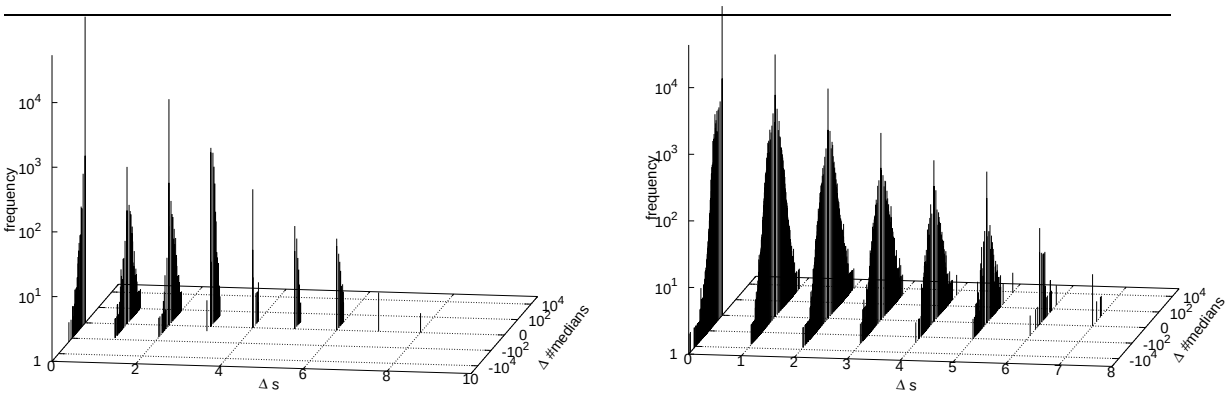


Figure 3.13 – Comparison of pIMP and IMP medians: score difference Δs and the difference of the size of the median sets Δ#medians relatively to the IMP solution; histogram for random data sets with $(n, r) \in R$ (left); histogram for the biological data sets (right)

data sets TCIP is more than 1 000 times faster when computing all medians (respectively one median).

The run time of TCIP is influenced to a large extent by the properties of the corresponding strong interval tree, as suggested by the theoretical results presented in Section 3.3.5. TCIP is especially fast for those data sets where many instances have no prime nodes or the quotient permutations defining the oIMP instances to be solved are small. These properties apply perfectly to the three *Chordata* groups and the *Hexapoda*. If the strong interval tree has only one large prime node, with quotient permutations of the size of the input permutations, it is expected that the run times of TCIP is about as large as the run time of Caprara's median solver. This is what can be observed for the groups containing the *Mollusca* and *Nematodes*, i.e. on average one prime node per instance with quotient permutations nearly as large as the input permutations and very similar run times.

Number of preserving medians

The number of preserving medians in the different groups of test instances of the biological data is shown in Table 3.8. It can be seen that the number of preserving medians can be very large. Nevertheless, the fraction of the overall computation time that is needed for the enumeration of all medians (after all oIMPs were solved) is very small in the artificial as well as in the biological data set, even when the number of preserving medians is very large: it is always either < 0.1 seconds or $< 1\%$ of the overall computation time. This is due to the fact that the prime node components for the test instances are very small.

	one median					all medians				
	s_{med}	s_{avg}	$s_{max}[\cdot 10^3]$	$g[\%]$	$g_{max}[s]$	s_{med}	s_{avg}	$s_{max}[\cdot 10^3]$	$g[\%]$	$g_{max}[s]$
act	9.8	88.8	2.4	0.46	0.00	106.8	675.2	16.5	0.34	-0.07
nac	74.8	2 141.0	1 356.6	0.05	0.00	694.8	13 917.3	8 074.6	0.02	0.00
cho	43.6	1230.7	1264.4	0.18	-0.03	390.1	9057.7	10630.8	0.13	-0.31
ehx	1.7	6.9	0.2	8.48	-1.70	1.8	30.0	2.1	10.91	-59.85
cru	1.9	137.8	21.2	9.82	-1.04	2.6	836.1	140.1	5.00	-23.76
hex	23.0	2 437.0	535.0	5.32	-144.32	175.1	12 172.6	2 341.5	4.51	-2 043.07
cmy	2.4	125.1	33.2	13.88	-16.37	2.9	691.4	207.8	12.00	-256.52
art	3.2	1 077.5	1 360.7	9.33	-464.60	4.6	3 727.9	3 405.1	7.71	-2 031.26
abm	1.2	28.9	10.0	18.18	-792.27	1.3	135.2	44.4	11.95	-3 263.20
np	1.1	22.2	1.4	26.96	-548.20	1.1	82.8	6.3	13.62	-1 256.39
cp	11.2	13.3	0.1	0.00	0.00	23.0	33.4	0.3	0.00	0.00
sim	1.9	2.7	0.2	13.57	-17.15	2.7	4.0	1.4	3.37	-237.19

Table 3.7 – Table showing the median (s_{med}), average (s_{avg}), and maximum (s_{max}) values of the speedup, i.e. the run time of Caprara's median solver divided by the run time of TCIP, for the biological data sets (rows act–cp) and the 1 000 simulated data sets for each combination of (n, r) (shown in row sim); note that the thousandth part of s_{max} is given; furthermore the percentage of the data sets where Caprara's median solver is faster (g) and the maximum difference (g_{max} in seconds) of the run times for the data sets where Caprara's median solver is faster

	avg	(max)
act	6.14	496
nac	4.47	328
cho	5.89	2 596
ehx	156.14	3 709
cru	59.86	4 167
hex	58.98	128 684
cmy	105.30	20 853
art	93.42	128 684
abm	178.67	8 023
np	1 491.94	70 004
cp	46.87	1 841
sim	7.38	9 848

Table 3.8 – Average and maximum of the number of preserving medians in the instances of the different groups of the biological data set and for the simulated data set (row sim)

Comparison of pIMP and IMP medians

In Figure 3.13 the relation of solutions for the IMP and the corresponding pIMP is given. For each of the artificial and biological test instances the score difference Δs and the difference of the size of the median sets Δ#medians relatively to the IMP solution is computed. A positive value for Δs (respectively Δ#medians) indicates that the pIMP solution has a larger score (respectively the median set of the pIMP solution is larger). For each combination of Δs and Δ#medians the number of instances that occurred in the corresponding class has been computed. The histograms of the number of instances that fall into a certain class are given in Figure 3.13 for the 61 999 artificial test instances with $(n, r) \in R$ and the 132 407 biological test instances, which have been solved by Caprara's median solver and TCIP. For $\Delta s = 0$ the size of median sets can only decrease, as in this case each median of a pIMP instance is a median for the corresponding IMP instance, too. For larger values of Δs the median set of a pIMP instance can have more or less elements than the median set of the corresponding IMP instance. Figure 3.13 clearly shows that preserving medians are good approximations for the corresponding IMP. This can be seen on the basis of the 55 768 (i.e. 90.0%) of all artificial test instances and 109 551 (i.e. 82.7%) of all instances have a score difference of $\Delta s \leq 1$. The score of the pIMP is never larger than 2.5 times the score of the IMP solution for the simulated data sets. For the biological data sets the fraction is always smaller than 1.59. Consequently, TCIP can also be regarded as an efficient heuristic algorithm for the IMP with a good optimisation behaviour.

Specific aspects of TCIP

Computing one median with TCIP is much faster than computing all medians (see Figure 3.12 and Table 3.7). For example, for the 1 762 solved data sets of the abm group, the average speedup per instance is 22.9 when computing only one median with TCIP compared to computing all medians with TCIP. For some instances of the art group even speedups of more than 900 were achieved. As shown in the theoretical part of this chapter, the reasons are threefold i) in the induced oIMPs only one median has to be computed, ii) due to Proposition 3.3.15 the number of oIMPs to be solved can reduced, and iii) the preserving medians do not have to be enumerated.

If only one instead of all preserving medians are computed, the number of oIMP instances is potentially reduced. This reduction strongly depends on the number of 3-signs in a prime node I with a prime node parent, for which a minimal $\xi(I, s)$ is found (compare Proposition 3.3.15). Let Ξ be the number of 3-signs leading to such a minimal $\xi_{\min}(I, s)$ (i.e. Ξ is the multiplicity of $\min_{s\in\Sigma}\{\xi_{\min}(I, s)\}$ in the multiset $\{\xi_{\min}(I, s) : s \in \Sigma\}$). In Figure 3.14 the frequency of $\Xi = 1, \ldots, \Xi = 4$ is depicted for all prime nodes with a prime node parent i) in all random test instances and ii) in all biological test instances. Obviously $\Xi = 1$ occurs very often.

In the random data set this is the case in 542 of the 640 prime nodes with prime node parents. Due to small values for Ξ, in the random data set the number of oIMPs to be solved is reduced from 64 067 to 62 214, i.e. 1 853 oIMPs do not have to be solved due to Proposition 3.3.15. In the biological data set the number of oIMPs to be solved is reduced from 133 865 to 129 402. The number of saved oIMPs seems relatively small. This is because prime node components are very small in all test instances, e.g. for 99 677 of the biological data sets no oIMP can possibly be saved because only one oIMP is to be solved. Nevertheless, each oIMP is an NP-hard problem. Note that theoretically for prime nodes with prime node children for which $\Xi = 1$ holds, the number of saved oIMPs is exponential in the number of such prime node children.

3.5 Conclusion

The preserving inversion median problem was introduced. This is the inversion median problem with the additional — biologically motivated — constraint that gene groups have to be preserved. This is the gene groups present in the given gene orders are present in a preserving inversion median, too. Additionally it is required that the applied inversion do not destroys a gene group. Common intervals have been used for determining gene groups. It has been shown empirically that common intervals often occur for mitochondrial gene orders of most taxa and many common intervals are destroyed when the inversion median problem is solved without considering common intervals.

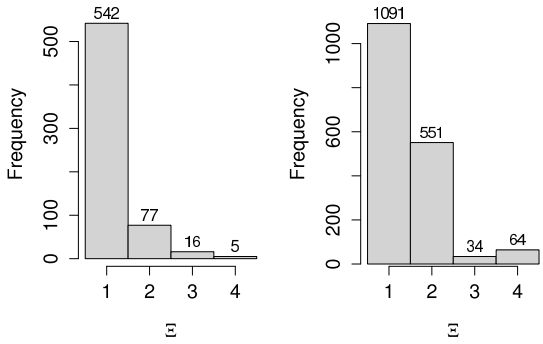


Figure 3.14 – Frequency of different multiplicities Ξ of 3-signs $s \in \Sigma$ leading to a minimal $\xi_{\min}(I, s)$ in prime nodes with prime node parents in, i) all artificial test instances with $(n, r) \in R$ (left) and, ii) all biological test instances (right)

Several different exact algorithm have been proposed for solving the preserving inversion median problem.

The algorithms CIP and ECIP are based on Caprara's branch and bound median solver. The algorithms cut the branch and bound search tree such that only preserving solutions are generated. In the case of CIP the cut is applied in the last step, i.e. it is checked if a generated permutation is preserving. For ECIP more elaborated techniques are used to avoid the enumeration of non preserving solutions. It was shown that both algorithms have reasonable run times for simulated as well as biological data sets. Due to the improvements introduced in algorithm ECIP a speedup when compared to CIP was observed for the biological data sets.

The algorithm TCIP is based on an extension of strong interval trees, namely k-signed strong interval trees. It was analysed theoretically how the difficulty of the pIMP depends on the structure of the corresponding k-signed strong interval tree. Based on these results, algorithm TCIP was designed such that it can solve a pIMP instance by solving several smaller instances of the inversion median problem. It was shown empirically that often no IMP instances have to be solved at all, leading to a linear run time behaviour for TCIP. For computing one preserving median instead of all, several non-trivial techniques are applied within algorithm TCIP to increase the efficiency. The theoretical results in Bernt et al. (2008b) given only for $k = 3$ are presented here for the general case. For data sets of mitochondrial gene orders and for randomly generated data sets, several important properties of the corresponding strong interval trees have been analysed. Moreover, it was shown empirically that preserving median scenarios can be computed even faster with TCIP than standard (i.e. not necessarily preserving) median scenarios can be computed with the state of the art IMP solver by Caprara. This is remarkable because even computing the preserving minimum inversion distance between two gene orders is an NP-hard problem (whereas the

same problem with not necessarily preserving inversions can be solved in polynomial time). Furthermore, it was shown that algorithm TCIP can be used as a good heuristic for the IMP.

4 Shifting the focus from inversions to more general models of rearrangement

Phylogenetic hypothesis are often supported by the computation of parsimonious scenarios of genome-wide rearrangement operations. Especially mitochondrial gene orders became a very fruitful source for such investigations as the number of genes is not too large and for more than 1 000 species the mitochondrial gene order is known. In the literature inversions and transpositions are the most often considered genomic rearrangement operation for phylogenetic reconstruction (Blanchette et al., 1999; Sankoff, 1992). Even when only inversions and a small number of gene orders are considered, recovering a most parsimonious scenario is usually NP-complete (e.g. Caprara (2003); Tannier et al. (2009)). Considering combinations of rearrangement operations in event based reconstruction methods is done rarely (e.g. Bader et al., 2008).

In recent biological studies it was shown that the so called tandem duplication random loss (TDRL) operation is a genomic rearrangement operation that can be found several times in the mitochondrial gene order evolution, e.g. in millipedes (Lavrov et al., 2002) and deep-sea gulper eels (Inoue et al., 2003). A TDRL duplicates a continuous segment of genes, followed by the loss of one copy of each of the redundant genes. Another rearrangement pattern found several times in mitochondrial gene order evolution is the inverse transposition (Black 4th and Roehrdanz, 1998; Boore, 1999; Dávila et al., 2005; Miklós and Hein, 2005; Woo et al., 2007). That is the transposed part is inverted. But note, it is unclear if this operation is an atomic operation or the inversion and transposition appeared separately during the gene order evolution.

For the study of the gene order evolution of mitochondrial gene orders, all relevant rearrangement operations should be regarded, i.e. inversions, transpositions, inverse transpositions, and TDRLs.

In Section 4.2 algorithm `CREx` (common interval rearrangement explorer) is presented. Also `CREx` is based on the idea to preserve common intervals in order to preserve gene groups during evolution. This is achieved by using the strong interval tree data structure reflecting the properties of continuous gene groups. `CREx` infers heuristically a rearrangement scenario between two gene orders. This is achieved by the identification of patterns

in the strong interval tree corresponding to the considered genomic rearrangement operations. All four rearrangement types, which are important for reconstructing mitochondrial gene order evolution, are considered by `CREx`. With this set of operations `CREx` is well suited for studying mitochondrial gene order evolution. `CREx` has been implemented in C++. Algorithm `CREx`, a tutorial, and several detailed examples are available online at `http://pacosy.informatik.uni-leipzig.de/crex`. Based on pairwise inspection of the gene orders `CREx` can be used to manually infer the evolution of mitochondrial gene orders for given phylogenetic hypotheses (Perseke et al., 2008).

Such a manual analysis is very laborious. In Section 4.3 the algorithm `TreeREx` (tree rearrangement explorer) is proposed for automatising this task. `TreeREx` takes as input a binary, rooted phylogenetic tree and the gene orders of the taxa at the leaves and heuristically infers the corresponding rearrangement operations on the edges of the tree and the ancestral gene arrangements at the inner nodes of the tree. Algorithm `TreeREx` utilises the pairwise comparisons computed with algorithm `CREx`. The applicability of `TreeREx` is shown on several biological examples.

Section 4.5 presents a simple approach for studying a set of gene arrangements without a given phylogenetic tree. This approach is based on the insights gained from a large simulation study analysing the quality of `CREx`'s reconstruction which is presented in Section 4.4. The new approach facilitates the exploration of the rearrangement history of large gene arrangement data sets. In Section 4.5.2 the results of the approach for the complete mitochondrial gene order data set are presented.

4.1 Basic definitions

Recall the formal definitions of (signed) permutations and the four rearrangement operations relevant for `CREx` given in Section 2.1.2.

Common intervals and strong interval trees are used compliant to Chapter 2 and Section 3.3. Nevertheless, these definitions should be recalled here (for references see Chapter 2).

Let Π be a set of signed permutations of size n. In this section Π will contain two permutations, without loss of generality, $\Pi = \{\pi, \iota\}$. Thus, in the following only π will be specified because ι is given implicitly. A *common interval* of π is a subset of $\{1, 2, \ldots, n\}$ that is an interval in π (and in ι). Two intervals *overlap* if they have a non empty intersection and one is not included in the other. If two intervals do not overlap they *commute*. A common interval which commutes with every common interval is called a *strong common interval*. The tree with nodes corresponding to the strong common intervals and the edges defined by the minimal inclusion relation of the strong common intervals is the *strong interval tree* $T(\pi)$. The root node of $T(\pi)$ is the interval containing all elements and the leaves are the singletons. The *quotient permutation*s of a node in $T(\pi)$ is defined as the order of the child

intervals in one permutation relative to the other permutation. That is the permutation of the interval order in ι found in π. If the quotient permutation of a node is the identity permutation (respectively the inverse of the identity $(n \ \ldots \ 1)$) the node is called *linear increasing* (*linear decreasing*). For a linear increasing (decreasing) node the strong common intervals corresponding to children appear in the same (opposite) order in the input gene orders. If the quotient permutation is not *linear*, i.e. neither linear increasing nor decreasing, it is called *prime*. The type of linear nodes in the strong interval is indicated with sign + for linear increasing nodes and − for linear decreasing nodes. Prime nodes inherit the sign of the parent iff the parent node is linear. The *signed quotient permutation* of a node is the quotient permutation of the node where each node is signed according to the sign of the corresponding child node. The set of all strong common intervals as well as the signed strong interval tree can be computed in time $\mathcal{O}(kn)$ for k signed permutations of size n (Bergeron et al., 2008a).

Note, the strong interval tree as defined here correspond to 2-signed strong interval trees in Chapter 3. The first sign is omitted because it is always + for the given permutations.

4.2 CREx

4.2.1 The basic algorithm

CREx (Bernt et al., 2008a, 2007c) is an algorithm to heuristically determine parsimonious preserving rearrangement scenarios for pairs of unichromosomal genomes. The considered rearrangement operations are inversions, transpositions, inverse transpositions, and tandem duplication random loss. The algorithm uses the fact that each of the four rearrangement operations leads to a pattern in the strong interval tree. To illustrate this each of the four rearrangement operations is applied to the identity permutation and the resulting strong interval tree is computed. Figure 4.1 shows the applied rearrangement operations and the resulting strong interval trees. More formally, the following patterns appear for the different operations when applied to a permutation π.

- If an inversion $\rho_{\mathrm{I}}(i, j)$ is applied, a linear node with a linear parent node of opposite sign occurs in the corresponding strong interval tree. The linear node reflects the common interval of all elements that are inverted. This pattern was used before to find preserving inversion scenarios (see Bérard et al. (2007) and Section 3.3).

- If a transposition $\rho_{\mathrm{T}}(i, j, k)$ is applied, the corresponding strong interval tree has a linear node with elements $\{\pi(i), \ldots, \pi(k)\}$ that has two linear children reflecting the transposed common intervals $\{\pi(i), \ldots, \pi(j)\}$ and $\{\pi(j+1), \ldots, \pi(k)\}$. The sign of the node is different from the signs of the child nodes.

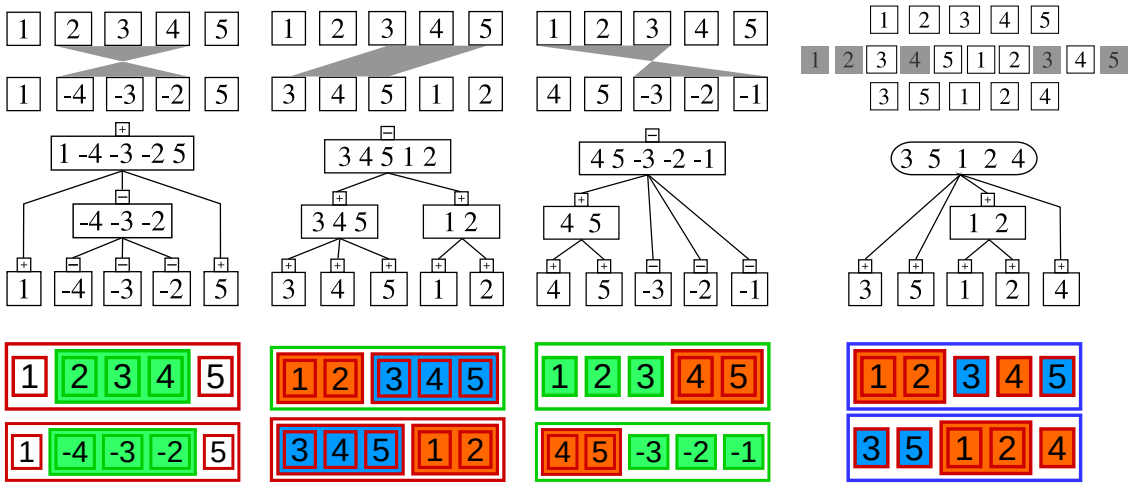


Figure 4.1 – Genomic rearrangement events considered in `CREx`; from left to right: inversion, transposition, inverse transposition, and tandem duplication random loss; top: the effects of an example rearrangement on the identity permutation of length five; middle: strong interval tree of the resulting permutation and the identity permutation; bottom: alternative representation of the strong interval tree as *family diagram* (shown with respect to identity and with respect to the resulting permutation); each box represents a strong common interval; the inclusion relation is represented by the inclusion of the boxes; the line colours of the boxes indicate node types (red: increasing, green: decreasing, and blue prime); the fill colours of the boxes indicate rearranged intervals (inversion: green, transposition: a red and a blue box mark the swapped intervals, inverse transposition: the inverse transposed interval is shown in green and the transposed part in red, TDRL: intervals kept in the first copy in blue and intervals kept in the second copy in red)

- If an inverse transposition $\rho_{\text{iT}}(i, j, k)$ is applied, the corresponding strong interval tree has a linear node with elements $\{\pi(i), \ldots, \pi(k)\}$. One child is a linear node reflecting the common interval of elements $\{\pi(i), \ldots, \pi(j)\}$ which are not inverted due to the inverse transposition. This child must have a different sign as its parent. The other elements that are involved in the inverse transposition are singletons that are children of node $\{\pi(i), \ldots, \pi(k)\}$, which must have the same sign as their parent.

- A tandem duplication loss operation ρ_{TDRL} leads to a prime node which includes all the elements involved in the rearrangement operation. The children of the prime node correspond to the maximal intervals of the permutation which have been kept in the same copy.

Algorithm `CREx` computes for two input permutations π and σ the strong interval tree for these permutations. Then `CREx` searches for patterns corresponding to rearrangement operations. If a pattern is identified, the corresponding rearrangement operation ρ is included in the scenario and the next pattern is searched in the strong interval tree of $\pi \circ \rho$ and σ

(the pattern for ρ will not occur in this strong interval tree). This process is repeated until a complete scenario is inferred. Remark, the nodes of the strong interval tree are commuting by definition. Therefore, rearrangement operations inferred from different nodes of the strong interval tree commute.

4.2.2 Handling of prime nodes

Special care has to be taken when prime nodes occur in the strong interval tree. In algorithm CREx a prime node is an indicator for one or several TDRLs. As a TDRL operation does not change the orientation of the involved elements, inversions are utilised to equalise the signs of the elements in a prime node's quotient permutation. The methods to achieve this have been developed in Ramsch (2007). Algorithm CREx uses a heuristic approach to identify a parsimonious number of inversions and TDRLs for the corresponding prime node. Let π be the signed quotient permutation of a prime node which has to be transformed into the identity permutation (without loss of generality). Two variants are now included in the latest version of CREx to infer the necessary inversions:

- A set of inversions is applied to the origin permutation π to make all elements positive, and then, starting from the resulting permutation, a parsimonious TDRL scenario to ι is computed. (*inversions-first*)

- First, a set of inversions is applied to ι, such that all the signs of the elements become equal to the signs of the elements of π, resulting in a permutation π'. Then a parsimonious TDRL scenario from π to permutation π' is computed. (*inversions-last*)

A parsimonious TDRL scenario is computed with the algorithm from (Chaudhuri et al., 2006) in both methods. In the following the method to compute the inversions-first scenarios is described. Let π be a signed permutation — the signed quotient permutation of a prime node — and ι the identity permutation of the same length. First, inversions have to be applied to π such that all elements of the resulting permutations have the same sign as the corresponding elements in ι, i.e. they have to be positive. Let $B^-(\pi) = (b_1, \ldots, b_k)$ be the maximal intervals of π consisting of elements that have a negative sign and $B^+(\pi) = (c_1, \ldots, c_{k-1})$ be the maximal intervals of π consisting of elements that have a positive sign which are in between the intervals of B^-, i.e. b_i, c_i, b_{i+1} are adjacent in π, for $1 \leq i < k-1$. Note, there may be elements left of b_1 or right of b_k that are not included in any of the intervals. Furthermore, let (s_i, e_i), with $i \in [1:k]$ be the pair of start and end indices of the i-th interval of B^-. Every inversion $\rho_{\text{I}}(s_i, e_j)$, with $1 \leq i \leq j \leq k$, reduces the number of intervals in $B^-(\pi)$ by one. This is because the elements of every interval b_l, with $i \leq l \leq j$, and every interval in between, i.e. c_l with $i \leq l < j$, switch the sign.

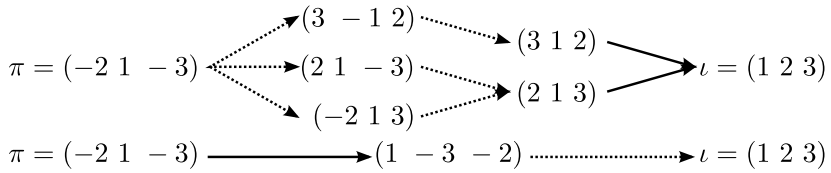


Figure 4.2 – Computation of combined inversion and TDRL scenarios for permutation π = (-2 1 -3) to ι; top: inversions-first; bottom: inversions-last; arrows represent a TDRL and dotted arrows inversions

The application of inversions of this kind leads after k steps to a permutation consisting of only positive elements. In each step of the described method with k maximal negative intervals there are $\binom{k}{2}$ possible inversions. The exhaustive enumeration leads to a set of permutations consisting of positive elements only. From this set the subset of permutations having a minimal TDRL distance to ι is selected. This strategy is exact, i.e. a shortest scenario consisting of inversions followed by TDRLs is computed.

The computations of inversions-last scenarios is very similar, but the definition of the intervals B^- and B^+ has to be generalised. Let $B^-(\pi, \sigma) = (b_1, \ldots, b_k)$ be the maximal intervals of π consisting of elements that have a different sign in σ and $B^+(\pi, \sigma) = (c_1, \ldots, c_{k-1})$ be the maximal intervals of π consisting of elements that have the same sign in σ which are in between intervals of B^+. Note, when σ is the identity, these definitions match the definitions of $B^-(\pi)$ and $B^+(\pi)$ given above. For the computation of the inversions-last scenarios, $B^-(\iota, \pi)$ and $B^+(\iota, \pi)$ are considered. Similar as above the application of an inversion $\rho_{\mathrm{I}}(s_i, e_j)$ to ι, with $1 \leq i \leq j \leq k$, reduces the number of maximal intervals with different signs by one. Thus, after the application of k inversions to ι, a permutation π' is reached whose elements have the same sign as the elements of π. For the permutations π and π' the TDRL distance can be computed because all elements have the same sign. The exhaustive generation of all possible inversion scenarios of length k for ι equalising the signs with respect to π leads to a set of permutations. From this set the permutations those permutations are selected which can be reached from π with a minimum number of TDRLs.

Non parsimonious inversions-first or inversions-last scenarios are discarded.

Example 4.2.1. *Consider the permutation* π = (-2 1 -3). *For the computation of the inversions-first scenarios (see Figure 4.2) the two maximum negative intervals* $B^-(\pi)$ *are given by* $((1,1),(3,3))$. *Thus, there are three possible inversions which reduce the number intervals in* $B^-(\pi)$ *by one: the inversion of the complete permutation* ($\rho_I(1,3)$) *and the inversions of the single elements* 3 *and* 2 ($\rho_I(1,1), \rho_I(3,3)$). *This results in a set of three permutations with a single sign difference with respect to* ι. *The subsequent inversion of this element leads to two possible permutations with positive elements only. Both of these permu-*

tations can be transformed with a single TDRL (actually transpositions) into ι. Thus, the shortest inversions-first scenarios consists of two inversions and a single TDRL.

Figure 4.2 shows also the computation of the inversions-last scenarios. The identity permutation ι has, with respect to π, two elements with a different sign. These are the elements 2 and 3 which form an interval in ι. Thus, there is only a single maximum interval in $B^-(\iota, \pi) = ((2, 3))$. The inversion of the interval leads to the permutation (1 -3 -2) whose elements have, with respect to π, the same signs. The input permutation π can be transformed with a single TDRL into (1 -3 -2) (which is a transposition). Thus, the shortest inversions-last scenario consists of a single inversion and a single TDRL.

The inversions-first scenario is rejected because it is not parsimonious in the number of rearrangements.

Note, even if the number of possible inversions that can be applied at every step is $\mathcal{O}(k^2)$ and the number of steps is k, the number of different possible parsimonious scenarios grows exponentially with k. Algorithm `CREx` uses a simple brute force approach and each possible shortest inversion scenario is tested. This strategy leads to lots of recomputation because the intermediate permutations on the scenarios to positive permutations can be reached by different combinations of inversions applied in the algorithm. This is prevented by storing intermediate permutations, generated during the run of the algorithm, in a tree based data structure, which is used to check if a permutation was processed before.

The experimental results in Section 4.4 will show that long combined inversion and TDRL scenarios are often of limited reliability. In order to avoid long run times `CREx` implements an option to limit the number of considered inversions-first (-last) scenarios. That is the search is aborted as soon as a user specified number of scenarios with the same score are found.

Connected components of prime nodes may occur in the strong interval trees. For the computation of a parsimonious rearrangement scenario it would be necessary to check all possible sign assignments to the prime nodes of the connected components, similar as described in Bérard et al. (2007) for preserving inversion scenarios or in TCIP described in Section 3.3.4. This is not done here in favour of an efficient algorithm. Furthermore, combined inversion and TDRL scenarios are not guaranteed to be parsimonious, as a mixed sequence of inversions and TDRLs may result in a smaller scenario. Remark, the TDRL scenario computed with the algorithm of Chaudhuri et al. (2006) may not be unique. This problem is treated theoretically and practically in Chapter 5.

4.2.3 Pattern search order

Some care has to be taken with regard to the search order of the four patterns due to the following reasons. If `CREx` would search for inversion patterns before transposition patterns, it would find three inversion patterns instead of the transposition. The reason for this is that

the inversion pattern is included in the transposition pattern. This corresponds to the fact that every transposition can be replaced by three inversions. In fact, if the strong interval tree has no prime nodes, the inversions found with the inversion patterns are sufficient to explain the differences of the corresponding permutation and the resulting scenario is a parsimonious preserving inversion scenario (Bérard et al., 2007). Similar ambiguities occur:

- if inversions are identified before inverse transpositions (every inverse transposition can be replaced by two inversions),
- if inversion and inverse transposition patterns are identified before transposition patterns (every transposition can be replaced by an inversion and a transposition), or
- if transposition and inversion patterns are identified before inverse transpositions (every inverse transposition can be replaced by an inversion and a transposition).

Therefore, the search order for the patterns of the genomic rearrangement operations is defined to be i) transpositions, ii) inverse transpositions, iii) inversions, and iv) combined TDRL and inversion scenarios for prime nodes. This order introduces a bias favouring those operations which are searched first. One method to resolve this bias is described in section Section 4.2.5.

4.2.4 Pseudo code and run time

Algorithm 5: `CREx`

Input: Two signed permutations π and σ
Output: A preserving rearrangement scenario transforming π into σ

```
1  Compute the strong interval tree T(π, σ);
2  foreach node I of T(π, σ) in post order do
3      if I is linear then
4          FindTranspositions(I);
5          if transposition pattern does not match then
6              FindInverseTranspositions(I);
7          if transposition and inverse transposition pattern do not match then
8              foreach child J of I do
9                  FindInversions(J);
10             if I is root node then
11                 FindInversions(I);
12     else
13         FindTDRLs(I);
```

The pseudo code of algorithm CREx is given in Algorithm 5. Given two permutations of length n CREx starts with the construction of the strong interval tree. This can be accomplished in linear time (Bérard et al., 2007; Bergeron et al., 2008a). Recall, a strong interval tree has $\mathcal{O}(n)$ nodes and a node has $\mathcal{O}(n)$ neighbour nodes. In the following it is assumed that the data structure allows constant time random access to the neighbours of a node and their properties.

For each of the nodes of the strong interval tree the patterns for the four considered rearrangement operations are identified in the order given in Section 4.2.3. The pattern identification is accomplished in by the functions FindTranspositions, FindInverseTranspositions, FindInversions, and FindTDRLs. The TDRL pattern matches only for prime nodes and the patterns of the other considered rearrangements can only match at linear nodes. If a pattern matches, the other patterns do not have to be checked. The inversion pattern is checked delayed, i.e. the inversion pattern is checked for the child nodes of the current node (and for the root node). This has to be done because the transposition pattern and inverse transposition pattern incorporate the signs of the child nodes in contrast to the inversion pattern. For example, consider a strong interval tree containing a transposition pattern at a node I. In the post order traversal of the tree the inversion pattern would match for each of the children of the node I. This prevents that the transposition pattern is identified at node I.

In order to identify the patterns, the signs of adjacent nodes, i.e. the parent node and the child nodes, have to be examined. The run time for checking if a pattern matches for a node are as follows:

- For the inversion pattern a single comparison of the sign of the node and the sign of the parent is sufficient. Thus, this check needs time $\mathcal{O}(1)$.

- The check for the transposition pattern also needs time $\mathcal{O}(1)$ because it has to be checked if the node has two children and the signs of the node have to be compared with the sign of the parent as well as with the signs of the two children .

- The check if the inverse transposition pattern matches needs time $\mathcal{O}(n)$ because the sign of the node has to be compared to the sign of the parent and signs of all child nodes.

- For the TDRL pattern it is sufficient to check if the node is a prime node and therefore the run time of the check is $\mathcal{O}(1)$. For the computation of the actual rearrangements implied by the node, two cases have to be considered:
  - If all elements of the quotient permutation have the same sign then the TDRL scenario is computed with the linear time algorithm from Chaudhuri et al. (2006),

– otherwise the brute force algorithm described in Section 4.2.2 is applied, which has run time that is in the worst case exponential in the number of maximal intervals with a different sign.

For the rearrangements identified for linear nodes, i.e. inversions, transpositions, and inverse transpositions, as well as prime nodes with a quotient permutation that has no negative elements, the rearrangement is appended to the rearrangement scenario. A rearrangement may affect $\mathcal{O}(n)$ elements. Note, Algorithm `CREx` does not modify the strong interval tree. That is after the identification of a rearrangement ρ the tree is not updated to $\mathcal{T}(\pi \circ \rho)$, but instead the tree $\mathcal{T}(\pi)$ is used throughout the algorithm. This is valid because the nodes are traversed in post order. That is a rearrangement, identified in the algorithm, does not modify the patterns in the strong interval tree examined at later stages of the algorithm.

Therefore, `CREx` has run time $\mathcal{O}(n^2)$ if the strong interval tree has no prime nodes or the quotient permutations of prime nodes have only positive signs. For prime nodes whose quotient permutation have positive and negative elements the combined inversion and TDRL identification method is used (see Section 4.2.2). This method has an exponential run time behaviour and therefore clearly dominates the run time. In Section 4.5 it is shown that real world (mitochondrial) data sets have often strong interval trees with only linear nodes.

4.2.5 Alternative scenarios

The order of the pattern identification introduces a bias in the identification of the rearrangements favouring operations which are identified first. Therefore, `CREx` computes not only the identified rearrangements but also possible alternative scenarios.

There are several possibilities to add alternative scenarios to the operations identified by the patterns in the strong interval trees. `CREx` uses the following. Let S and T be two sets of elements.

- Three alternative scenarios are added to a transposition $\rho_{\mathrm{T}}(\{S, T\})$:
  - three inversions $\rho_{\mathrm{I}}(S)$, $\rho_{\mathrm{I}}(T)$, and $\rho_{\mathrm{I}}(S \cup T)$;
  - the inverse transposition $\rho_{\mathrm{iT}}((S, T))$ and the inversion $\rho_{\mathrm{I}}(S)$; and
  - the inverse transposition $\rho_{\mathrm{iT}}((T, S))$ and the inversion $\rho_{\mathrm{I}}(T)$.
- Two alternative scenarios are added to an inverse transposition $\rho_{\mathrm{iT}}((S, T))$:
  - the inversion $\rho_{\mathrm{I}}(S)$ and the transposition $\rho_{\mathrm{T}}(S, T)$;
  - the inversions $\rho_{\mathrm{I}}(T)$ and $\rho_{\mathrm{I}}(S \cup T)$.

Additionally, parsimonious inversions-first or inversion-last scenarios identified for prime nodes (see Section 4.2.2) are stored as alternative scenarios.

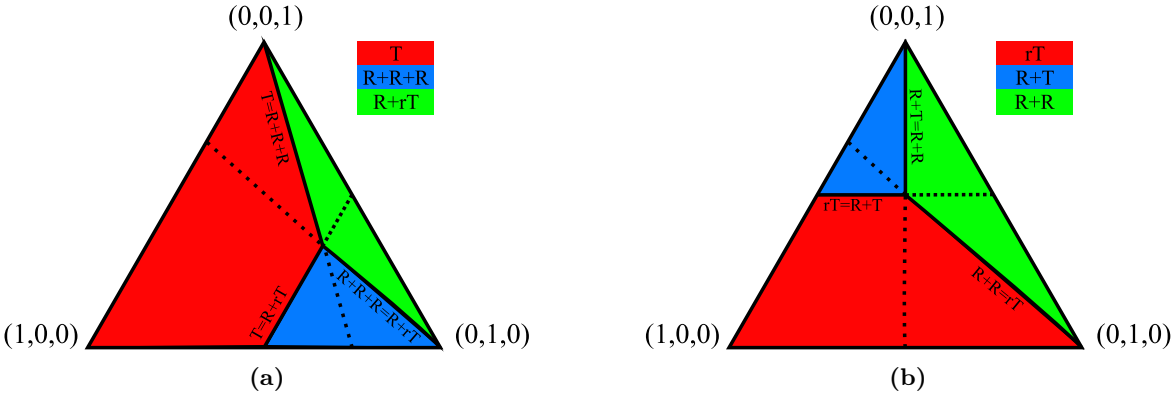


Figure 4.3 – Simplex plot showing the influence of different ratios of the weights in normalised weighting schemes (w_I, w_T, w_{iT}) on the selection of alternative scenarios for a) alternatives of a transposition and b) alternatives of an inverse transposition; different colours visualise which alternative is the most parsimonious; weighting schemes for which two alternatives are equally parsimonious are shown as lines; at the point where the three lines intersect all alternatives are equally parsimonious

4.2.6 Weighting

By introducing alternative scenarios the problem appears how to choose between the different alternatives. One obvious possibility is to give a weight to each rearrangement operation and to choose alternatives such that only scenarios with a smallest sum of weights are returned. This is discussed in the following.

By introducing weights $\mathcal{W} = (w_I, w_T, w_{iT}, w_{TDRL})$, with $0 < w_I, w_T, w_{iT}, w_{TDRL}$, for the different rearrangement types, it is possible to choose between alternatives in a post processing step or by discarding the generation of certain alternatives. The influence of different weights on the choice of the most parsimonious alternative(s) is depicted in Figure 4.3. Consider the alternative scenarios for transpositions and inverse transpositions introduced above. For $\mathcal{W}$, with $w_I : w_T : w_{iT} = \frac{1}{6} : \frac{1}{2} : \frac{1}{3}$, the alternatives of a transposition and for $\mathcal{W}$, with $w_I : w_T : w_{iT} = \frac{1}{4} : \frac{1}{4} : \frac{1}{2}$, the alternatives of an inverse transposition are equally parsimonious. Furthermore, there are several weighting schemes where two of the three alternatives are equally likely, e.g. for $w_T = w_I + w_{iT}$ the inversion plus inverse transposition costs the same as the transposition. There is no weighting scheme for which all three alternatives of inverse transpositions as well as transposition are equally parsimonious, at least one of the alternatives is inferior. The original rearrangement is parsimonious if all operations have the same weight, i.e. all alternatives are cancelled out. Thus, the generation of certain alternatives by Algorithm `CREx` could be easily ruled out. But up to now this has not been implemented in `CREx`.

The choice of weights is a difficult task that is crucial for the reliability of the reconstructions. A method to discard alternatives by using a phylogenetic tree, is presented in the next section.

4.3 TreeREx

Although algorithm CREx supports a user to find parsimonious rearrangement scenarios for a given phylogenetic hypothesis with more than two input genomes, this process has to be done by manual inspection of pairwise scenarios. The overlap of pairwise scenarios for different pairs of taxa can be utilised to infer events on the edges of a given phylogenetic tree. This has been done for a data set of mitochondrial gene orders of *Echinoderm* species (Perseke et al., 2008). Algorithm TreeREx (tree rearrangement explorer) automates this procedure. A given phylogenetic tree is analysed in a bottom-up manner by iteratively considering triples and quadruples of gene orders. TreeREx utilises the pairwise comparisons suggested by CREx, assigns genomic rearrangement events to edges of the phylogenetic tree, and computes the permutations assigned to ancestral nodes.

4.3.1 Consistency

Let $\Pi = \{\pi_1, \ldots, \pi_m\}$ be signed input permutations and $T = (V, E)$ be a binary phylogenetic tree with the permutations $\pi_1, \ldots, \pi_m$ assigned to the leaf nodes $v_1, \ldots, v_m$. Let $r(\pi_i, \pi_j)$ denote a pairwise rearrangement scenario between π_i and π_j, $1 \leq i, j \leq m$. For TreeREx the pairwise scenarios are inferred with the CREx algorithm described in Section 4.2. For the moment assume that the rearrangements in scenarios are pairwise commutative — as it is the case for rearrangement scenarios inferred by CREx except when prime nodes have to be explained by more than one operation. Also the possibility to enrich the CREx scenarios with alternatives is not considered for the moment. How non commutative and alternative scenarios are processed in TreeREx is described in Section 4.3.3.

Let π be the permutation to be assigned to the parent node v of two siblings v_i and v_j. Let r_ϵ, $\epsilon \in \{i, j\}$ be the inferred rearrangement events on edge $(v, v_\epsilon) \in E$ by intersecting all pairwise scenarios from any permutation towards π_ϵ. Formally r_ϵ is computed by

$$r_\epsilon = \bigcap_{\pi_l \in \Pi \setminus \{\pi_\epsilon\}} r(\pi_l, \pi_\epsilon), \epsilon \in \{i, j\} \tag{4.1}$$

If the rearrangements in the scenarios are commutative and do not contain alternatives, the rearrangement scenarios can be regarded as a set. Thus, the intersection is well defined in this case.

Figure 4.4 – Illustration of the consistent reconstruction of the ancestral permutation of the permutations π_i and π_j in a tree with three leaves; the considered pairwise CREx scenarios are shown as dashed lines; intersections of the pairwise rearrangement scenarios are represented as grey shaded areas; left: reconstruction of the putative ancestral permutation of π_i; right: reconstruction of the putative ancestral permutation of π_j

The rearrangements r_ϵ imply a putative ancestral state π'_ϵ which can be computed by inversely applying r_ϵ to π_ϵ, i.e. $\pi'_\epsilon = \pi_\epsilon \circ r_\epsilon^{-1}$. The rearrangements have to be applied inversely because asymmetric rearrangements are considered. For symmetric rearrangement operations there is no difference between applying a rearrangements and the inverse application. The obtained rearrangement events are said to be *consistent* with tree T, if the inverse application of the rearrangements to the corresponding permutations leads to the same permutation. That is $\pi_i \circ r_i^{-1} = \pi_j \circ r_j^{-1} = \pi$ holds. An ancestral permutation that has been found with rearrangements that are consistent with tree T is also called consistent. For an illustration see Figure 4.4.

If some of the pairwise rearrangement scenarios are wrong, no consistent reconstruction can be found. This may result in an empty or incomplete intersection in Equation (4.1) and no consistent ancestral permutation can be found. It may still be possible to find the ancestral state if the wrong scenarios are disregarded, but it is unknown which scenarios are wrong. The decision if a rearrangement scenario is wrong can be based on a relaxed definition of consistency restricting the set of intersected scenarios in Equation (4.1). Suppose, the number of pairwise scenarios used in the intersection in Equation (4.1) is reduced by k, with $k < m$, for a subtree with m leaves. Let r'_i and r'_j be the potentially different scenarios resulting from these relaxed intersections. If the inverse application of r'_i to π_i and r'_j to π_j results in the same permutation π, i.e. if $\pi = \pi_i \circ r'_i = \pi_j \circ r'_j$, π is said to be *k-consistent* (with tree T). Obviously, a node is 0-consistent iff it is consistent. If no k can be found, such that a node is k-consistent, the node is *inconsistent*. It is possible that several k-consistent ancestral permutations exist. In this case TreeREx chooses a permutation occurring in the most k-consistent reconstructions.

Example 4.3.1. *Assume that the ancestral permutation π of π_1 and π_2 in the tree shown in Figure 4.5 can not be inferred consistently. One possible relaxation of the intersected scenarios by two scenarios is to discard $r(\pi_3, \pi_1)$ and $r(\pi_4, \pi_2)$ in the intersection. That is*

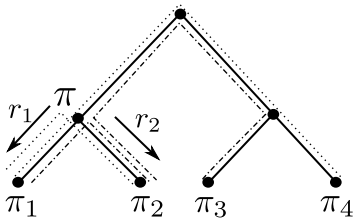


Figure 4.5 – Illustration of the intersections made for the 2-consistent reconstruction of the putative ancestral permutation π of π_1 and π_2 as described in Example 4.3.1; dashed lines connecting pairs of permutations depict the used pairwise scenarios; the two different line types indicate which scenarios are used to derive r_1 and which for r_2

$r_1 = r(\pi_2, \pi_1) \cap r(\pi_4, \pi_1)$ *and* $r_2 = r(\pi_1, \pi_2) \cap r(\pi_3, \pi_2)$. *If* $\pi = \pi_1 \circ r_1^{-1} = \pi_2 \circ r_2^{-1}$ *then* r_1 *and* r_2 *are* 2*-consistent (with* T*). Assume that the exclusion of* $r(\pi_3, \pi_1)$ *and* $r(\pi_4, \pi_1)$ *from the intersections also leads to the* 2*-consistent permutation* π*, the reduction of the intersections by* $r(\pi_4, \pi_1)$ *and* $r(\pi_3, \pi_2)$ *leads to a different* 2*-consistent ancestral permutation* π'*, and all other reduction by two scenarios do not lead to a consistent ancestral permutation,* ***TreeREx*** *chooses* π *to be the ancestral permutation. That is because* π *was reconstructed in the most* 2*-consistent cases.*

4.3.2 Method

Algorithm `TreeREx` traverses subtrees in a given (binary) phylogenetic tree in a bottom-up manner beginning with subtrees induced by the permutations assigned to leaf nodes. That is `TreeREx` iteratively selects a subtree with three or four leaf nodes, which have a height of two, and for which the permutations assigned to the leaf nodes of the subtree are known. For the inner nodes of the subtree of the leaf nodes of the subtree the ancestral permutations are computed and the next subtree is selected. This is repeated until all (but the root node) have an assigned permutation. More formally, `TreeREx` proceeds as follows. Let π_1 and π_2 be permutations assigned to two sibling nodes in the phylogenetic tree. Let π be an unknown permutation to be assigned to the parent node v of these siblings. Let π' be the permutation assigned to the sibling v' of v, and let π'_1 and π'_2 be the permutations assigned to the child nodes of v'. If the permutation of π' is known, `TreeREx` infers the permutation for π by utilising the subtree induced by π_1, π_2, and π'. If the permutation of v' is not known, `TreeREx` infers the permutation for π by utilising the subtree induced by π_1, π_2, π'_1, and π'_2. Due to the bottom-up traversal such an induced subtree with three or four permutations assigned to leaf nodes can always be found, as long as at least two inner nodes of the complete phylogenetic tree have no permutation assigned yet.

Assigning permutations to inner nodes and events to the edges of an induced subtree T_S is done as follows. In a first step `TreeREx` checks if consistent rearrangement scenarios and a consistent permutation can be found by utilising the necessary pairwise `CREx` scenarios of the leaves of T_S. If this fails, `TreeREx` tries to infer rearrangement events by iteratively checking, if a k-consistent permutation can be assigned to an inner node. That is k is increased from one to its maximal possible value or until a k-consistent permutation is found. In the case that no k-consistent permutation can be found for an inner node, a fallback strategy is applied as follows. For the scenario of π_1 and π_2 — computed by `CREx`— each possible ancestral permutation for π, based on each possible partition of the events in the scenario of π_1 and π_2, is computed. Let $\Gamma(\pi_1, \pi_2)$ be the set of these permutations and

$$\Gamma(v) = \begin{cases} \pi & \text{if } v \text{ has an assigned permutation } \pi \\ \Gamma(\pi_1, \pi_2) & \text{else.} \end{cases}$$

To choose an ancestral permutation for v, the putative ancestral permutations $\pi \in \Gamma(v)$ and the permutations in $\pi' \in \Gamma(v')$ determined for its sibling v' are taken into account. For each combination of potential ancestral permutations $\pi \in \Gamma(v)$ and $\pi' \in \Gamma(v')$ the set of rearrangement events is computed. π and π' are chosen, such that the sum of the (weighted) number of rearrangement events is minimal. Hence, a weighting function (denoted by q in the pseudo code of `TreeREx`) has to be defined. Currently only equal weights are implemented in `TreeREx`, i.e. the number of rearrangements is minimised. The pseudo code of algorithm `TreeREx` is given in Algorithm 6.

Algorithm `TreeREx` is designed to support biologist when analysing real biological data and aims at inferring biologically evident events. Therefor, the presented heuristic approach includes the four phylogenetic rearrangement operation known for mitochondrial genomes. The outcome of `TreeREx` includes the consistency of internal nodes, which are a good indicator for the support of the inferred events. Having only consistent internal nodes in a subtree strongly supports the inferred rearrangements and ancestral gene orders. Obviously, as even simplifications of the underlying problems are NP-complete and as the number of possible scenarios for two species can be immense, this trade-off between usability and optimality is needed. If the outcome includes inconsistent and a large sequence of rearrangement operations on one edge only, the support for this reconstruction is very small and a reexamination of the gene orders, the tree topology, or the `CREx` reconstructions is necessary. But, in contrast, if the outcome includes mainly k-consistent nodes for small values of k, the support for the inferred events is very strong.

Algorithm 6: `TreeREx`

Input: A phylogenetic binary tree $T = (V, E)$ with leaf nodes $\{\pi_1, \ldots, \pi_n\}$
Output: A mapping of phylogenetic events on edges and a mapping of permutations to internal nodes

while *($\exists$ induced subtree T_S of height 2 with 3 or 4 leaf nodes, for which permutations are assigned to leaf nodes only)* **do**
  Let $\Pi \leftarrow \{\pi_1, \ldots, \pi_m\}$ be the set of permutations assigned to the leaf nodes of T_S ($m = 3$ or $m = 4$ holds);
  Let π_i, π_j be permutations assigned to sibling leaf nodes of T_S, for which no permutation is assigned to their parent node v;
  Let v' be the sibling of v;
  `// Check if a consistent permutation can be inferred:`
  $r_i \leftarrow \bigcap_{\pi \in \Pi \setminus \pi_i}$ `CREx` (π, π_i);
  $r_j \leftarrow \bigcap_{\pi \in \Pi \setminus \pi_j}$ `CREx` (π, π_j);
  Let $\varpi_i \leftarrow \pi_i \circ r_i^{-1}$ be the permutation computed by inversely applying r_i to π_i;
  Let $\varpi_j \leftarrow \pi_j \circ r_j^{-1}$ be the permutation computed by inversely applying r_j to π_j;
  if *($\varpi_i = \varpi_j$)* **then**
    Assign all events r_i and r_j to the corresponding edges;
    Assign the permutation ϖ_i to v;
  else
    `// Check if a` k`-consistent permutation can be inferred:`
    $k \leftarrow 0$;
    while *(no k-consistent permutation was found) $\wedge$ (k is not maximal)* **do**
      $k \leftarrow k + 1$;
      Similar to the consistent case, try to infer
        i) k-consistent permutation to be assigned to v and
        ii) k-consistent events to the corresponding edges;
    `// Inconsistent case:`
    if *no permutation was assigned to node v* **then**
      Compute all possible ancestral permutations $\Gamma(v)$ for node v;
      Compute all possible ancestral permutations $\Gamma(v')$ for node v';
      `// Assign permutation to` v `in a parsimonious manner`
      forall *($\pi \in \Gamma(\pi_i, \pi_j)$ and $\pi' \in \Gamma(\pi_k, \pi_l)$)* **do**
        Compute the weighted number of events $q($`CREx` $(\pi, \pi'))$;
      Assign π to v , such that $q($`CREx` $(\pi, \pi'))$ is minimal;
      Assign the inferred events to the edges;

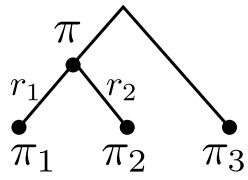


Figure 4.6 – Topology of the phylogenetic tree used in Example 4.3.2

4.3.3 Extensions

So far, algorithm TreeREx has been explained only in its basic version. Several extensions are used in order to improve the reliability of the inferred rearrangements and ancestral gene arrangements. This includes the handling of non-commutative rearrangement operations, the use of alternative scenarios for transpositions and inverse transpositions, the use of the direction information of TDRL events, and the inclusion of shared adjacency scenarios as additional alternative scenarios.

Ordered and alternative scenarios

The pairwise scenarios as computed by CREx are inferred by heuristically identifying patterns in the strong interval tree. So far it was assumed that there is only one pairwise scenario. CREx can enrich the computed rearrangement scenarios with alternatives for transpositions and inverse transpositions (see Section 4.2.5) and computes alternative inversion-first and inversion-last scenarios for prime nodes (compare Section 4.2.2). Furthermore, it was assumed so far, that events of a scenario can be applied in a commutative manner, which is only true if the strong interval tree has linear nodes only. The rearrangements inferred from a prime node can not be applied commutatively in general but are assumed to be in a certain order.

	CREx scenario	alternatives
$r(\pi_1, \pi_2)$	$\rho_{\mathrm{iT}}((\{3\},\{4,5\}))$	i) $\rho_{\mathrm{T}}(\{\{3\},\{4,5\}\}) \circ \rho_{\mathrm{I}}(\{3\})$ ii) $\rho_{\mathrm{I}}(\{4,5\}) \circ \rho_{\mathrm{I}}(\{3,4,5\})$
$r(\pi_1, \pi_3)$	$\rho_{\mathrm{T}}(\{\{1,2\},\{3,4,5\}\})$	i) $\rho_{\mathrm{I}}(\{1,2\}) \circ \rho_{\mathrm{I}}(\{3,4,5\}) \circ \rho_{\mathrm{I}}(\{1,2,3,4,5\})$ ii) $\rho_{\mathrm{I}}(\{3,4,5\}) \circ \rho_{\mathrm{iT}}((\{3,4,5\},\{1,2\}))$ iii) $\rho_{\mathrm{I}}(\{1,2\}) \circ \rho_{\mathrm{iT}}((\{1,2\},\{3,4,5\}))$
$r(\pi_2, \pi_3)$	$\rho_{\mathrm{I}}(\{4,5\}), \rho_{\mathrm{I}}(\{1,2\}), \rho_{\mathrm{I}}(\{1,\ldots,5\})$	

Table 4.1 – Rearrangement scenarios inferred by the CREx method for pairs of permutations from Example 4.3.2 and the alternative scenarios; note that all three rearrangement scenarios do commute, i.e. $r(\pi_i, \pi_j) = r(\pi_j, \pi_i)$ holds for $i, j \in [1:3], i \neq j$; $r(\pi_2, \pi_3)$ does not have an alternative scenario

Example 4.3.2. *With the following example it is demonstrated how alternative scenarios, as used in* **CREx**, *can help in the reconstruction of consistent ancestral states. Let* $\pi_1 = (2\ 1\ 5\ 4\ 3)$, $\pi_2 = (2\ 1\ -3\ 5\ 4)$, *and* $\pi_3 = (5\ 4\ 3\ 2\ 1)$. *Consider the problem of inferring the ancestral gene order* π *in a phylogenetic tree as given in Figure 4.6. The pairwise scenarios determined by the* **CREx** *method are given in Table 4.1. In order to reconstruct* π *the following intersections are determined:*

i) $r_1 = r(\pi^2, \pi^1) \cap r(\pi^3, \pi^1) = \emptyset$ *and*

ii) $r_2 = r(\pi^1, \pi^2) \cap r(\pi^3, \pi^2) = \emptyset$.

The inverse application of the intersections leads to an inconsistent state as $\pi^1 \circ r_1^{-1} \neq \pi^2 \circ r_2^{-1}$. *In this example no k-consistent ancestral permutation exists and the fallback method has to be applied.*

The following intersections are computed when the alternatives for the pairwise **CREx** *scenarios are included:*

i) $r_1 = r(\pi^2, \pi^1) \cap r(\pi^3, \pi^1) = \rho_I(\{3, 4, 5\})$ *and*

ii) $r_2 = r(\pi^1, \pi^2) \cap r(\pi^3, \pi^2) = \rho_I(\{4, 5\})$.

The inverse application of these intersections gives the same permutation (2 1 -3 -4 -5) *which is consistent with the given tree.*

Both alternative and ordered scenarios have to be handled properly when the intersection of `CREx` scenarios are computed according to Equation (4.1). Each alternative is handled separately, e.g. the intersection of two alternative scenarios is an alternative scenario consisting of the intersections of all-vs-all alternatives. The intersection of an ordered sequence of events is the largest common suffix shared with the other scenario. As a formal description is very technical, this is illustrated with a small example.

Example 4.3.3. *Let* $r_1 = \rho_{TDRL} \rightarrow \rho_{I2} \rightarrow \rho_{I1}$ *denote an ordered sequence of one TDRL and two inversions. Let* $r_2 = \rho_T || \{\rho_{I4}, \rho_{I5}, \rho_{I6}\}$ *denote an alternative of either a transposition* ρ_T *or three commuting inversions. Let* $r_3 = \rho_{I2} \rightarrow \rho_{I1}$ *and* $r_4 = \rho_T || \{\rho_{I4}, \rho_{I6}\}$ *denote two other sequences. Let* $r_1 || r_2$ *and* $r_3 || r_4$ *represent two pairs of alternative scenarios inferred by* **CREx**. *The intersection of these scenarios leads to the scenario represented by* $\rho_{I2} \rightarrow \rho_{I1} || (\rho_T || \{\rho_{I4}, \rho_{I6}\})$.

Handling of tandem duplication random loss events

When symmetric operations like inversions, transpositions, and inverse transpositions are considered, the assignment of the rearrangements separating two siblings in a phylogenetic tree to the edges towards their parent node can not be done without the use of additional (e.g. outgroup) information. Because of this, also the inference of the ancestral permutation is not possible without additional information. For example, let π, σ be two permutations that are separated by a single symmetric rearrangement s, i.e. $\pi \circ s = \sigma$ and $\sigma \circ s = \pi$. Then

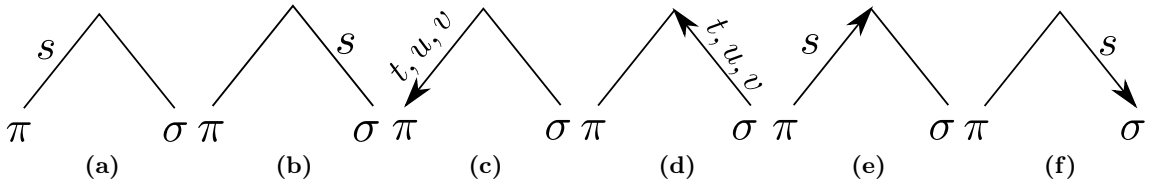


Figure 4.7 – Assignment of symmetric and asymmetric rearrangement operations to the edges of a phylogenetic tree; a,b) for a symmetric operation s both assignments are possible and parsimonious; c-e) for asymmetric operations only two assignments are possible with respect to the time information given by the rooted phylogenetic tree (c and f); only one of these possibilities is parsimonious (f)

it can not be decided if s is assigned to the edge from π to the ancestor or to the edge from σ to the ancestor (see Figures 4.7a and 4.7b). Note that using the outgroup information is exactly what `TreeREx` is doing automatically.

TDRLs are in general asymmetric operations (Chaudhuri et al., 2006). When a TDRL is applied to a permutation, more than one TDRL is necessary to get back to the original permutations. This holds for all TDRLs with the exception of transpositions which are included in the TDRL model. The asymmetry of the TDRL operation seems to be problematic because standard phylogenetic tools, e.g. distance matrix methods or easy approximations, can not be applied (Chaudhuri et al., 2006). But the asymmetry provides valuable phylogenetic information. If asymmetric operations, like TDRLs, are considered, the assignment to an edge can be accomplished without the use of outgroup information. The reason is that in a rooted phylogenetic tree nodes which are closer to the root can be interpreted as phylogenetically older. Hence, the edges of the tree imply a direction in the time.

For example, let π, σ be two permutations such that there is a single asymmetric rearrangement s with $\pi \circ s = \sigma$ and $\sigma \circ s \neq \pi$, i.e. the effect of s can not be reversed with the same rearrangement. Let t, u, v be a parsimonious scenario of three asymmetric operations which transform σ into π. Then only the option to assign s to the edge from the ancestor to σ is parsimonious and possible (see Figure 4.7f). This is because

i) assigning t, u, v to the edge from the ancestor to π is not parsimonious (Figure 4.7c),
ii) assigning s to the edge from π to the ancestor contradicts the time information given by the phylogenetic tree (Figure 4.7e), and
iii) assigning t, u, v (or a part of the scenario) to the edge from σ to the ancestor can be discarded because the direction of the rearrangements contradicts the time information given by a rooted phylogenetic tree (furthermore, this option is not parsimonious) (Figure 4.7d).

The particularities of the asymmetric TDRL operation are correctly realised by `TreeREx`. As mentioned in Section 4.3.1, special attention is spent on the inverse application of TDRLs

for checking the consistency of a permutation because TDRLs are not symmetric. Let π be a TDRL and σ the result of the application of a TDRL. For the application of a TDRL it is sufficient to store the information which elements are kept in the first and which elements are kept in the second copy. This information is not sufficient for the inverse application of a TDRL. Consider the following example. The tandem duplication of the whole permutation (1 2 3 4) and subsequent loss of $\{1, 3\}$ in the first copy and $\{2, 4\}$ leads unambiguously to (2 4 1 3). The inverse application of a TDRL can not be carried out unambiguously when only the information on which elements are kept in which copy is given. For example also the application of the TDRL to (1 3 2 4) results in (2 4 1 3). Therefore, the original permutation is stored for each TDRL to allow for an unambiguous inverse application.

Furthermore, `TreeREx` discards scenarios with TDRLs leading towards the root node and only takes the parsimonious alternative. For the k-consistent cases the correct behaviour is already implied by Equation (4.1) because the direction of the `CREx` scenarios is used. The consequence for the fallback method is to regard $\Gamma(\pi_2, \pi_1)$ additionally because it is insufficient to consider only $\Gamma(\pi_1, \pi_2)$ when TDRLs are included. Additionally, partitions of the scenarios which include TDRLs that contradict the time information of the phylogenetic tree are excluded.

Inclusion of shared adjacency scenarios

As presented so far, `CREx` infers only TDRLs and (if necessary) inversions for prime nodes. This is useful because a TDRL always leads to a prime node. But there are other possibilities leading to the emergence of a prime node, for example the accumulation of rearrangements — maybe due to a rearrangement hot spot in the corresponding interval of the permutation. Thus, explaining the differences represented by a prime node with TDRLs only is not adequate in such circumstances.

Therefore, a method presented in Zhao and Bourque (2007) was adapted to heuristically construct (ordered) scenarios consisting of transpositions and inversions. The basic idea is to identify inversions and transpositions by a proper analysis of adjacencies of the two input permutations. More precisely, let π and σ be two permutations of length n. The following rules are used for the inference of inversions and transpositions:

1. If σ has a pair of adjacencies $(\pi(i-1), -\pi(j))$ and $(-\pi(i), \pi(j+1))$, with $1 \leq i \leq j \leq n$, an inversion $\rho_{\mathrm{I}}(i, j)$ is inferred, and

2. if σ has a triple of adjacencies $(\pi(i-1), \pi(j+1))$, $(\pi(k), \pi(i))$, and $(\pi(j), \pi(k+1))$, with $1 < i \leq j < k < n$, a transposition $\rho_{\mathrm{T}}(i, j, k)$ is inferred.

The method analyses all pairs and triples of adjacencies of σ and infers a set of transpositions and inversions with the given rules. The application of the inferred rearrangements leads

to a set of permutations. A set of alternative ordered scenarios consisting of inversions and transpositions is obtained by the recursive application of the rule based approach to each permutation in the set.

This method is included in `TreeREx` and used for the inference of alternative scenarios for prime nodes. Note that also other approaches that may be more appropriate for the demands of certain data sets can be incorporated as additional alternative scenarios.

4.4 Results

In this section `CREx` and `TreeREx` are analysed on simulated as well as real, i.e. mitochondrial, data stets. In Section 4.4.1 the quality of the rearrangements reconstructed with `CREx` is analysed empirically in a large simulation utilising many different rearrangement models. Properties of the strong interval trees are analysed with respect to the reconstruction quality of `CREx`. Furthermore, special, biologically useful, simulation parameters are identified for which `CREx` gives high quality reconstructions. Section 4.4.2 presents a detailed example of the Algorithm `TreeREx` applied to a four *echinoderm* mitogenomes. A data set consisting of the mitogenomes of all *Echinodermata* and a large *teleost* mitochondrial genome data set are analysed with algorithm `TreeREx` in Sections 4.4.3 and 4.4.4.

4.4.1 Performance of `CREx` on simulated data

The quality of the results returned by `CREx` is analysed in a large study on simulated data for various models of genome rearrangement.

Each simulated data set is constructed by applying r random rearrangements to the identity permutation of length $n = 100$. The type of the rearrangements is determined with respect to a given probability vector $p = (p_{\mathrm{I}}, p_{\mathrm{T}}, p_{\mathrm{iT}}, p_{\mathrm{TDRL}})$ specifying the probabilities that a rearrangement is an inversion (p_{I}), a transposition (p_{T}), an inverse transposition (p_{iT}), or a TDRL (p_{TDRL}). For example $p = (1, 0, 0, 0)$ results in an inversion only rearrangement scenario, $p = (0.3, 0.3, 0.3, 0.1)$ gives a scenario where each rearrangement is with probability 0.3 an inversion, transposition, or inverse transposition and with probability 0.1 a TDRL. Random inversions (respectively transpositions and inverse transpositions) are chosen with equal probability from the set of all possible inversions (respectively transpositions and inverse transpositions). A random TDRL is generated choosing the duplicated interval at random and for each element it is chosen at random if it is kept in the first or second copy. The position of the elements to the left and right of the tandem duplicated interval are not altered.

The resulting permutation and the identity are used as input for `CREx` and the reconstructed rearrangement scenario is compared to the simulated scenario which generated the

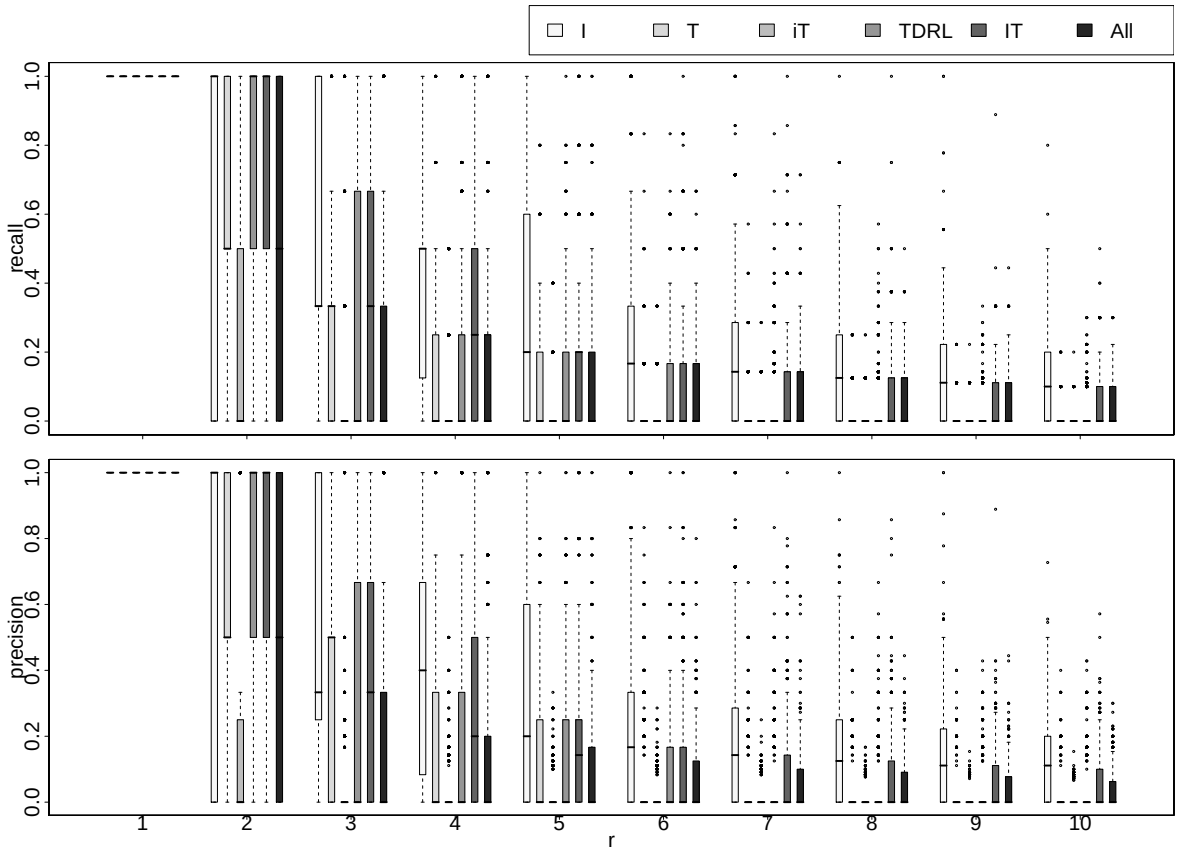


Figure 4.8 – Boxplots showing recall (top) and precision (bottom) for different rearrangement rates ($r \in [1:10]$) and different rearrangement models (indicated by grey scale); for each r from left to right: inversions only (I); transpositions only (T); inverse transpositions only (iT); TDRLs only (TDRL); equally likely inversions and transpositions with $p = (0.5, 0.5, 0, 0)$ (IT); mixed with $p = (0.3, 0.3, 0.3, 0.1)$ (All); for each combination of rearrangement rate r and rearrangement model 1 000 data sets have been simulated

permutation. Therefor the measures precision and recall are used. Let S be the simulated rearrangement scenario and C be the rearrangement scenario computed by `CREx`. Precision and recall are defined as $precision = \frac{|S \cap C|}{|C|}$ and $recall = \frac{|S \cap C|}{|S|}$, where the undefined cases are defined to be 1 if numerator and denominator are equal to 0 and 0 if only the denominator is equal to 0. Precision measures the exactness and recall the completeness of the reconstructed `CREx` scenarios. The intersection of the simulated and the rearrangement scenario is computed with the operation which is also used in `TreeREx`. Furthermore, the cardinality of a rearrangement scenario is computed as its length where the shortest alternative is considered.

Boxplots of precision and recall for different rearrangement rates r and rearrangement models are shown in Figure 4.8. Because the number of possibles outcomes for precision and

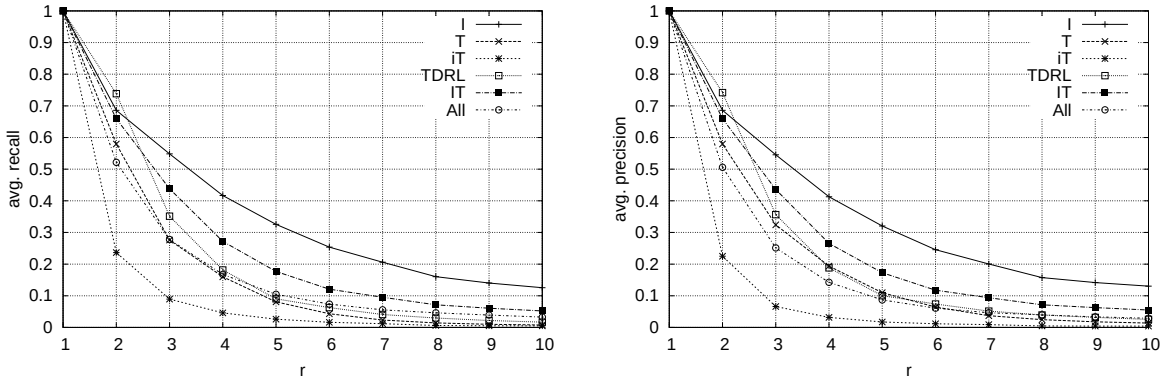


Figure 4.9 – Average values of recall (left) and precision (right) for different rearrangement rates ($r \in [1:10]$) and different rearrangement models; I, T, iT, TDRL, IT, and All as in Figure 4.8; for each combination of rearrangement rate r and rearrangement model 1 000 data sets have been simulated

recall is small boxplots are difficult to interpret, especially for small values of r. Therefore, the average values of precision and recall is shown separately in Figure 4.9.

For $r = 1$ the correct scenario was found by `CREx` for all data sets from all tested rearrangement models. This shows that the patterns used for the identification are defined and implemented correctly. But for larger values of r, the quality of the reconstructed scenarios decreases. The results are better when no transpositions or inverse transpositions are included in the rearrangement model. The majority of the rearrangement scenarios are correctly reconstructed for $r = 2$ (i.e. precision and recall are equal to one) in the inversion (respectively TDRL) only model, i.e. in 686 (710) data sets. Also for the rearrangement model, including inversions and transpositions with equal probabilities (respectively all types of rearrangements with $p = (0.3, 0.3, 0.3, 0.1)$), the correct scenario is still reconstructed for considerably more than one third of the data sets, i.e. in 425 (435), of the data sets. For the transposition only (respectively inverse transposition only) simulation model, the correct scenario is reconstructed for less than one third of the data sets, i.e. in 306 (respectively 211) cases. For $r > 9$ in the inversion only model, $r > 4$ in the transposition only model, $r > 3$ in the inverse transposition model, $r > 5$ in the TDRL only model, $r > 8$ in the model using inversions and transposition with equal probability, and $r > 5$ in the mixed model a perfect reconstruction (i.e. with a recall of one) could not be done for a single of the simulated data sets. But interestingly `CREx` was able to reconstruct at least a part of the simulated rearrangement scenario for many data sets. This observation is important because `TreeREx` may still work properly even if not the complete rearrangement scenario is correctly known. In the inversion only model a part of the rearrangement scenario, i.e. at least one rearrangement, could still be reconstructed correctly for a majority (694) of the data sets for

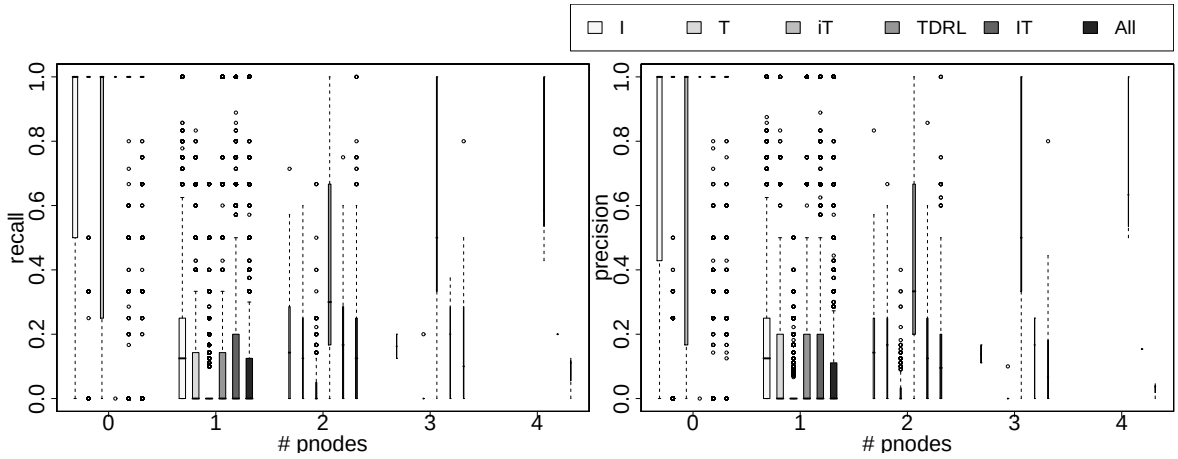


Figure 4.10 – Relation of the number of prime nodes p in the strong interval tree and recall (left) and precision (right) for different models of rearrangement; the boxplots summarise the results for 1 000 repetitions for each combination of rearrangement model and $r \in [1 : 10]$; I, T, iT, TDRL, IT, and All as in Figure 4.8

values of $r = 10$. For the data sets generated with equally likely inversions and transpositions (respectively transpositions only) some rearrangements are still identified correctly for the majority of the data sets for values of r up to five (respectively three). With respect to this criterion `CREx` showed the worst results for data sets simulated with the TDRLs respectively inverse transpositions, i.e. only for r up to two in the TDRL model and $r \leq 1$ in the inverse transposition only model, at least one rearrangement was reconstructed in the majority of the data sets.

Although these first results for simulated data seem not to be promising, criteria and additional biological constrained rearrangement models can be identified for which `CREx`'s performance is better.

The reconstruction quality depends on a large extent on the strong interval tree corresponding to the pair of permutations analysed by `CREx`. Before this is analysed in detail the amount of simulated data sets, which have at least one prime node in the corresponding strong interval tree, is given for the simulated data sets (see Table 4.2). With the exception of the TDRL and the mixed model of rearrangement no prime node is found in any of the data sets for $r = 1$. In the 155, cases where no prime node is found in the TDRL model, the method for the random generation of a TDRL produced a transposition (79 cases) or a rearrangement without any effect on the permutation (76 cases). The 80 cases with at least one prime node, found in the mixed rearrangement model, are caused by the same effect, i.e. in about 10% of the simulated data sets a TDRL is generated and in about 20% of these cases no real TDRL (i.e. a transposition or an empty rearrangement) is produced. With increasing values of r the number of data sets having prime nodes in the corresponding strong intervals

r	I	T	iT	TDRL	IT	All
1	0	0	0	845	0	80
2	0	574	425	981	399	477
3	248	912	843	1 000	714	812
4	538	987	978	1 000	902	947
5	766	1 000	994	1 000	971	986
6	876	1 000	999	1 000	995	998
7	954	1 000	1 000	1 000	997	1 000
8	979	1 000	1 000	1 000	999	1 000
9	992	1 000	1 000	1 000	1 000	1 000
10	999	1 000	1 000	1 000	1 000	1 000

Table 4.2 – Number of simulated data sets which have at least one prime node for different $r \in [1 : 10]$ and rearrangement models; 1 000 data sets per combination; I, T, iT, TDRL, IT, and All as in Figure 4.8

increases also. For $r > 5$ more than 99% of generated data sets have strong interval trees with at least one prime node in all rearrangement models excepting the inversions only model where only for $r \geq 9$ more than 99% of the data sets have a prime node.

Figure 4.10 shows precision and recall for the simulated data sets in relation to the number of prime nodes in the corresponding strong interval trees. If the strong interval tree of a pair of permutations has no prime nodes, the results of `CREx` for this pair are mostly perfect for all tested rearrangement rates $r \in [1 : 10]$. That is for 8 101 ($\approx 75\%$) of 10 833 random data sets without a prime node precision as well as recall is equal to one. For 9 616 ($\approx 89\%$) of the prime node free data sets precision and recall is greater than zero. Note that precision and recall values greater than zero in the TDRL model with zero prime nodes correspond to the seldom cases where a random TDRL is actually a transposition or a rearrangement without any consequence. These are correctly identified by `CREx`. The results of `CREx` for the simulated data sets with prime nodes in the corresponding strong interval trees clearly have much worse precision and recall. The `CREx` results have precision and recall of one for only 2 128 ($\approx 4\%$) of the 49 167 data sets with at least one prime node and for only 17 741 ($\approx 36\%$) of the data sets precision and recall are greater than zero. Data sets with $r = 1$ are always reconstructed correctly with `CREx` and have mostly a prime node free strong interval tree. Thus, the presented results may be biased by data sets with $r = 1$. But, the absence of prime nodes is still a good indicator of the quality of the `CREx` reconstruction when data sets with $r = 1$ are not considered. That is for 3 026, i,e. $\approx 53\%$, (respectively 4 541, i.e. $\approx 79\%$) of the 5 758 prime node free data sets with $r > 1$ precision and recall are equal to one (respectively greater than zero). In contrast, the `CREx` results have precision as well as

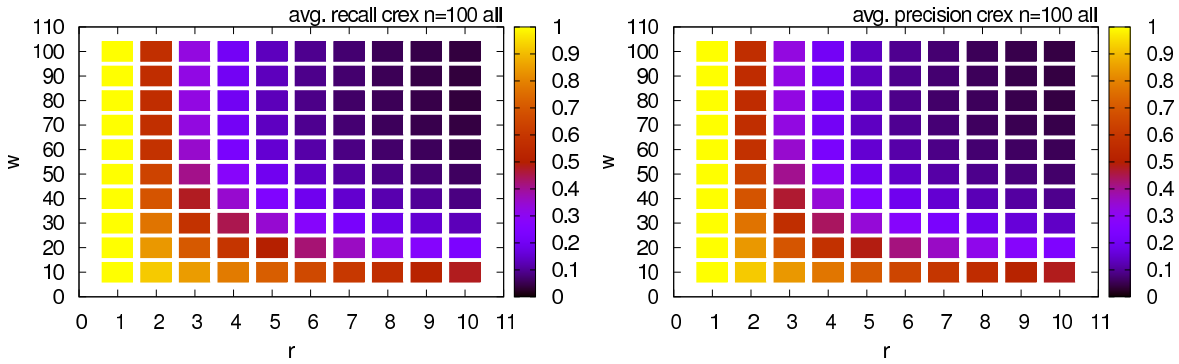


Figure 4.11 – Performance of `CREx` for simulated data sets ($n = 100$) for different numbers of rearrangements ($r = [1 : 10]$) and numbers of affected elements ($w = \{10, 20, \ldots, 100\}$); left: average recall; right: average precision; for each combination of r and w, 1 000 data sets have been simulated for each rearrangement models (I, T, iT, TDRL, IT, All); averages are computed including the results of all rearrangement models for each combination of r and w

recall of one for only 1 203, i.e. $\approx 7\%$, (respectively 16 816, i.e. $\approx 35\%$ with precision and recall greater zero) of the 48 242 data sets with at least one prime node and $r > 1$.

The presented results clearly show that the absence of prime nodes is a good indicator for the quality of the rearrangement scenario reconstructed by `CREx`.

In real world biological scenarios the rearrangements are not uniformly random but certain rearrangements appear with a higher probability than others. For example in circular genomes the region around the replication origin tends to be involved in rearrangements more frequently (see Suyama and Bork (2001) and for mitochondrial gene orders Fonseca and Harris (2008) and references therein), rearrangements in bacterial chromosomes are reported to be often symmetric around the origin and terminus of replicating (Eisen et al., 2000; Tillier and Collins, 2000), or short rearrangements are found more often (see Lefebvre et al. (2003) and references therein). Recently Darling et al. (2008), confirmed all three mechanisms in bacterial genomes.

In the following the influence of the length of the random rearrangements on the reconstruction fidelity of `CREx` is analysed. Therefor, the simulation method was modified such that only rearrangements affecting the order of at most w genes are allowed. From this set of allowed rearrangements, r randomly selected rearrangements (again with respect to a probability vector p) are applied starting from the identity permutation of length $n = 100$. The tested rearrangement size limits are $w \in \{10, 20, \ldots, 100\}$, where $w = 100$ is the unrestricted case that has already been presented above. The tested rearrangement models and rearrangement rates r are the same as above. For each combination of r and w 1 000 data sets have been simulated, i.e. 100 000 for each combination of the six considered rearrangement models.

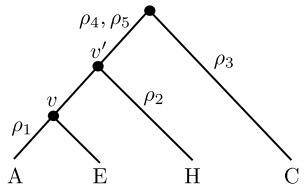


Figure 4.12 – Phylogenetic tree as used in Section 4.4.2; genomic rearrangement operations inferred by algorithm `TreeREx` are denoted by $\rho_1, \ldots, \rho_5$; inner nodes for which `TreeREx` inferred ancestral permutations are denoted by v and v'

The average values of precision and recall for the tested combinations of w and r are shown in Figure 4.11. Obviously, the results of `CREx` are of better quality for smaller values of w. For $r = 10$ the average of precision and recall increases from 0.04 for the unrestricted case, i.e. $w = 100$, to 0.48 for $w = 10$. For $r = 5$ the average of precision and recall for $w = 10$ is increased by 0.58 compared to the average values for $w = 100$. Thus, it was shown empirically that for certain, biologically more relevant, rearrangement models the `CREx` results are of much better quality compared to a model where the rearrangements are chosen randomly.

Note that the effects of increased reconstruction quality in the case of prime node absence and smaller rearrangements are not independent. For smaller rearrangements the number of simulated data sets without prime nodes increases, e.g. 53 819 of the data sets for $w > 50$ and 88 558 for $w \leq 50$ are prime node free (out of 300 000 for each w). This is in agreement with Sankoff (2002) where it was reported that short inversions lead to more preserved permutations but shuffled clusters of genes .

The computation of all 600 000 data sets, needed 21 minutes and 54 seconds (including the time for the evaluation of the returned scenarios) on a laptop with a 2.0 GHz processor. That is one rearrangement scenario was computed in about 10^{-3} seconds on average.

4.4.2 A detailed small biological example

The phylogeny of the *Echinodermata* has been investigated intensely (e.g. Littlewood et al. (1997)), but is still heavily discussed (Scouras et al., 2004). In order to exemplify the functionality of `TreeREx` and to show the practical usefulness of `TreeREx`, a small biological example of mitochondrial gene orders of four *echinoderm* species is used. That is

(A) the gene order shared by *Asteroidea*

```
CO1 R NAD4L CO2 K ATP8 ATP6 CO3 -S2 NAD3 NAD4 H S1 NAD5 -NAD6 CYTB F 12S E T
-16S -NAD2 -I -NAD1 -L2 -G -Y D -M V -C -W A -L1 -N Q -P
```

(E) the gene order common to *Echinoidea*,

```
CO1 R NAD4L CO2 K ATP8 ATP6 CO3 -S2 NAD3 NAD4 H S1 NAD5 -NAD6 CYTB F 12S E T P
-Q N L1 -A W C -V M -D Y G L2 NAD1 I NAD2 16S
```

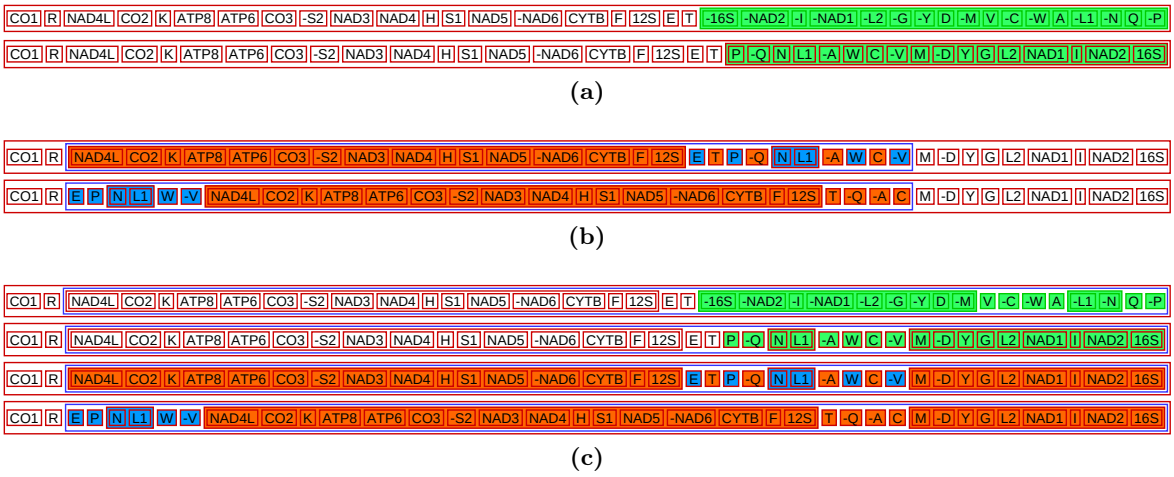


Figure 4.13 – The rearrangements inferred by Algorithm `CREx` for the pairwise comparisons in the subtree $((A, E), H)$ of the phylogenetic tree as used in Section 4.4.2; for each rearrangement the family diagram of the strong interval tree before and after the application of the rearrangement is shown (see Figure 4.1); a) the inversion ρ_1 that separates `A` and `E`; b) the TDRL ρ_2 from `E` to `H`; c) the rearrangement scenario from `A` to `H` consisting of an inversion (shown in the two lines on the top) leading to the configuration of `E` and a TDRL (shown in the two lines on the bottom)

(H) the gene order of *Holothuroidea C. miniata*, and
`CO1 R E P N L1 W -V NAD4L CO2 K ATP8 ATP6 CO3 -S2 NAD3 NAD4 H S1 NAD5 -NAD6 CYTB F 12S T -Q -A C M -D Y G L2 NAD1 I NAD2 16S`

(C) the gene order shared by the *crinoid* species *F. serratissima* and *P. Gracilis*
`CO1 R NAD4L CO2 K ATP8 ATP6 CO3 -S2 NAD3 NAD4 H S1 NAD5 -NAD6 CYTB P -Q N L1 -A W C -V M -D -T -E -12S -F -L2 -G -16S -Y -NAD2 -I -NAD1`

The gene orders are in agreement with the corrections described in Perseke et al. (2008) and the improved data set described in Appendix A. The favoured hypothesis in (Perseke et al., 2008) shown in Figure 4.12 is used as topology for the example. The *Ophiuroidea* are excluded from the example in order to keep it small, but see Section 4.4.3. `TreeREx` traverses the tree in a bottom up manner as follows.

i) The first subtree to be analysed is given by $((\mathtt{A}, \mathtt{E}), \mathtt{H})$. Let v denote the parent of `A` and `E`. The sibling of v is a node which has already an assigned permutation (permutation `H`). A single inversion ρ_1 of 17 genes (`(-16S ... -P)`) is predicted by `CREx` as rearrangement scenario that transforms `E` into `A` (see Figure 4.13a). Note, this scenario is symmetric, i.e. $\mathtt{CREx}(\mathtt{A}, \mathtt{E}) = \mathtt{CREx}(\mathtt{E}, \mathtt{A}) = \{\rho_1\}$. Without the outgroup information, given by `H` in this subtree, it is impossible to decide if the inversion ρ_1 happened on the edge $(v, \mathtt{E})$ and the permutation `A` is the ancestral permutation at v or the inversion happened on the

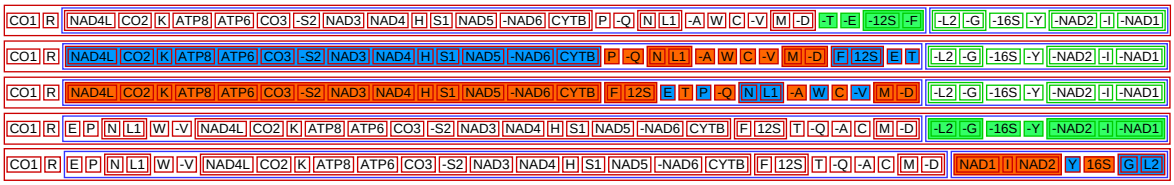
edge $(v, \texttt{A})$ and the permutation E is the ancestral permutation at v. The comparison of E and H with CREx returns a TDRL, i.e. $\texttt{CREx}(\texttt{E}, \texttt{H}) = \{\rho_2\}$ (see Figure 4.13b). Note that this scenario is not symmetric, i.e. the TDRL must be located on the edge $(v', \texttt{H})$. The scenario $\texttt{CREx}(\texttt{H}, \texttt{E})$ contains three TDRLs and is discarded because it is not parsimonious. Two events are predicted by CREx for the evolution between A and H, i.e. $\texttt{CREx}(\texttt{A}, \texttt{H}) = (\rho_1, \rho_2)$, with ρ_1 being the same inversion as found in $\texttt{CREx}(\texttt{A}, \texttt{E})$ and ρ_2 being the TDRL towards H as found in $\texttt{CREx}(\texttt{E}, \texttt{H})$ (Figure 4.13c). Also in this case the scenario in the other direction can be discarded, i.e. $\texttt{CREx}(\texttt{H}, \texttt{A})$ because it contains inversion ρ_1 and three TDRLs. For the computation of the rearrangement scenario from A to H the inversions-first procedure of CREx was used. The scenario has to be treated as ordered. This is because the inversion ρ_1 and the TDRL ρ_2 overlap.

The intersections defines the predicted events on the edges $(v, \texttt{A})$ and $(v, \texttt{E})$, i.e. $r(v, \texttt{A}) = \texttt{CREx}(\texttt{E}, \texttt{A}) \cap \texttt{CREx}(\texttt{H}, \texttt{A}) = \{\rho_1\}$ and $r((v, \texttt{E})) = \texttt{CREx}(\texttt{A}, \texttt{E}) \cap \texttt{CREx}(\texttt{H}, \texttt{E}) = \emptyset$. Applying ρ_1 inversely to A leads to the same permutation as applying no event to E. That is a consistent case and therefore the ancestor of A and E is E, which is assigned to node v. Furthermore, event ρ_1 is assigned to the edge (v, A).

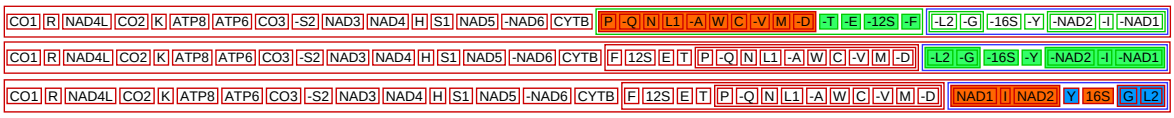
ii) The second subtree analysed by TreeREx is $((v, \texttt{H}), \texttt{C})$, with E assigned to v. Let v' denote the parent of v and H (see Figure 4.12). The ancestral permutation at node v' and the rearrangements on the edges to v and H are to be determined. The sibling of v' is the leaf node C and has therefore an assigned permutation. In order to determine the rearrangements on the edge from v' to H, TreeREx determines the intersection of the scenarios $\texttt{CREx}(v, \texttt{H}) = \texttt{CREx}(\texttt{E}, \texttt{H}) = \{\rho_2\}$ (see Figure 4.13b) and $\texttt{CREx}(\texttt{C}, \texttt{H})$. $\texttt{CREx}(\texttt{C}, \texttt{H})$ also includes the TDRL ρ_2 (see Figure 4.14a) and therefore the intersection is $r(v', \texttt{H}) = \{\rho_2\}$.

The comparison $\texttt{CREx}(\texttt{H}, v) = \texttt{CREx}(\texttt{H}, \texttt{E})$ gives three TDRLs (which are the non parsimonious counterpart of ρ_2). This scenario has an empty intersection with $\texttt{CREx}(\texttt{C}, \texttt{E})$ (see Figure 4.14b). Thus, $r(v', v) = \emptyset$. As applying ρ_2 inversely to H gives the same permutation as applying no operation to E, v' is also consistent and permutation E is also assigned to node v'.

iii) To infer the operations on the two edges incident to the root node, the pairwise scenario of C and E is computed. $\texttt{CREx}(\texttt{C}, \texttt{E}) = \{\rho_3, \rho_4, \rho_5\}$, with ρ_3 being an inverse transposition as shown in Figure 4.14b, ρ_4 being an inversion of (-L2 ... -NAD1), and ρ_5 being a TDRL event as shown in Figure 4.14b. $\texttt{CREx}(\texttt{E}, \texttt{C})$ leads to more than three events (not shown), as the TDRL ρ_5 is replaced with two transpositions. Therefore, TreeREx assigns ρ_3, ρ_4, and ρ_5 to the edges incident to the root node. As ρ_5 is a TDRL, it has to be on the edge



(a)



(b)

Figure 4.14 – The rearrangements inferred by Algorithm `CREx` for the subtree $((v, \mathtt{H}), \mathtt{C})$; for each rearrangement the family diagram for the configuration before the application is shown; a) the pairwise comparisons of `C` to `H`: an inversion, a transposition (note that both together have the same effect as ρ_3), the TDRL ρ_2, the inversion ρ_4, and the TDRL ρ_5; b) `C` to `E` the inverse transposition ρ_3, the inversion ρ_4 and the TDRL ρ_5

towards v'. Without an outgroup it is impossible to determine on which edge ρ_3 and ρ_4 occurred.

Two more notes on the scenario from `H` to `C` as shown in Figure 4.14a. `CREx` returns an inversion of the genes T, E, 12S, and F and a transposition involving the same four genes (see Figure 4.14a). `CREx` can not identify this rearrangement as an inverse transposition because the elements are part of a prime node which are explained by inversions or TDRLs in `CREx`. The alternative is the inverse transposition ρ_3 as found in the scenario from `H` to C (Figure 4.14a). Thus, if `TreeREx` would continue the bottom-up traversal with more outgroup gene orders, an intersection of these two scenarios results in the inversion and the transposition. Furthermore note, the transposition is identified as a part of the TDRL scenario which is computed with the algorithm of Chaudhuri et al. (2006). Nevertheless, `CREx` reports this correctly as a transposition.

4.4.3 *Echinodermata*

In Section 4.4.2 the gene order of four *Echinodermata* was used to describe algorithm `TreeREx`. In this subsection all *echinoderm* gene orders from Perseke et al. (2008) are utilised for the analysis of `TreeREx`. The phylogenetic tree for this echinoderm data set has been obtained by a careful analysis of the mitochondrial protein sequences in Perseke et al. (2008). The operations inferred by `TreeREx` are depicted in Figure 4.15a. In Perseke et al. (2008) the same results were found for this tree by manual inspection of pairwise `CREx` scenarios. None of the ancestral permutations was inferred inconsistently, and only two permutations were k-consistent with $k > 0$. The TDRL separating *C. miniata* from the ancestral gene order of

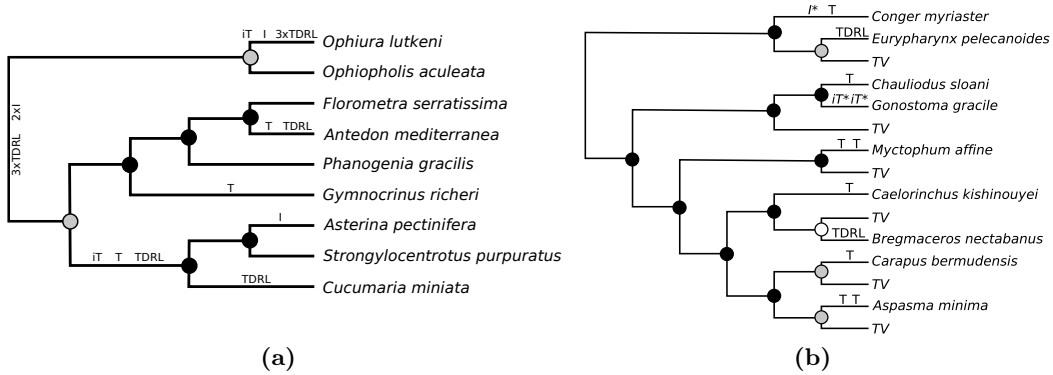


Figure 4.15 – Rearrangement events inferred by algorithm `TreeREx`; a) *Echinodermata* (see Section 4.4.3), the four gene orders used in Section 4.4.2 are represented here as (A): *A. pectinifera*, (E) *S. purpuratus*, (H): *C. miniata*, and (C): *G. richeri*; b) *Teleostei* (see Section 4.4.4); abbreviations used: I=inversion, T=transposition, iT=inverse transposition, TDRL=tandem duplication random loss; rearrangements with "*" are reconstructed differently in the literature (see text); nodes indicate consistency: black node=consistent, grey node=k-consistent with $k > 0$, white node=inconsistent; TV stands for species that have the typical vertebrate gene order

Echinoidea is discussed in Arndt and Smith (1998). The ancestral state of *Ophiuroidea* is very difficult to infer, as the available gene orders are heavily rearranged (also in Scouras et al. (2004) the ancestral state of *Ophiuroidea* remained unresolved). Although algorithm `TreeREx` infers an ancestral permutation, it utilises a sequence of several genomic rearrangement operations including three TDRLs. This is not very likely because three TDRLs are half of the diameter of the TDRL distance.

from a biological point of view. Yet, if the *Ophiuroidea* are not considered, the resulting operations have a strong support, and interestingly all four rearrangement operations considered in this chapter are necessary to explain the evolutionary history. The computation time of `TreeREx` was 0.1 seconds for this data set on a laptop with 2.0 GHz processor.

Note, meanwhile two new *Holothuroidea* gene orders can be found in the GenBank which are not included in the data set. One of these two gene orders is equal to the gene order of *C. miniata* and the other is equal to the gene order of the *Echinoidea*. The tree topology used here would imply that the *Holothuroidea* are not monophyletic. But assuming an alternative tree topology, which was already discussed and analysed with `CREx` in Perseke et al. (2008), the gene order differences can be explained with monophyletic *Holothuroidea* and the same number of rearrangements.

4.4.4 *Teleostei*

For the analysis of the teleost mitochondrial genomes the phylogenies suggested in Miya et al. (2003) and Inoue et al. (2003) have been merged. The mitochondrial gene orders of the corresponding species have been taken from the improved data set described in Appendix A. Most of the gene orders are identical to the typical vertebrate gene order (Miya et al., 2001, 2003), denoted by '*TV*' in the following, which is common to most vertebrate mitochondrial genomes, e.g. the gene order found on human mitochondrial genomes. Therefore, subtrees with identical gene order have been collapsed to a single leaf node carrying the *TV* gene order. Note, running `TreeREx` with the uncollapsed tree as input gives the same result, i.e. no rearrangements are predicted in subtrees where all leaves have the *TV* gene order.

In the data set are nine teleost mitogenomes with a gene order that differs from the *TV* order. These are *Chauliodus sloani*, *Myctophum affine*, and *Caelorinchus kishinouyei* (Miya et al., 2001); *Carapus bermudensis*, *Bregmaceros nectabanus*, *Aspasma minima* (Miya et al., 2003); the gene order shared by *Sigmops gracile* and *Gonostoma gracile* (Miya and Nishida, 1999); *Conger myriaster* (Inoue et al., 2001); and *Eurypharynx pelecanoides* (Inoue et al., 2003). Note, there are two slight differences compared to the data set used in Bernt et al. (2008a), i.e. the gene order of the mitochondrial genome of *G. gracile* is added and the tRNA-Thr is inverted in the new gene order of *C. myriaster*. *Diaphus splendidus* could still not be included in this study because the tRNA-Pro is missing in the annotation.

The result of the `TreeREx` analysis is depicted in Figure 4.15b. Apparently all but one of the ancestral gene orders were inferred consistent or k-consistent, $k \leq 1$. The only inconsistency occurred for the ancestral permutation of *B. nectabanus*. The cause for all three nodes that are inferred not consistently are overlapping rearrangements that mislead the pairwise comparisons with `CREx`. The TDRL towards *E. pelecanoides* overlaps with the transposition and the inversion that is inferred by `CREx` for the comparison of *TV* and *C. myriaster*. Similarly, the transposition separating *TV* and *C. bermudensis* overlaps with both transpositions that separate *TV* and *A. minima*. The reason for the single inconsistently inferred node are overlapping rearrangements, too. In this case the TDRL towards *B. nectabanus* and the transposition on the edge to *C. kishinouyei*. Nevertheless, the subtree of the corresponding three species has a consistent root node, and can be explained with only one transposition and one TDRL. Therefore, it can be concluded that the genomic rearrangement operations found by `TreeREx` are very likely. The TDRL towards *B. nectabanus* is depicted in Figure 4.16. Note that the difference to the k-consistently inferred ancestor of *E. pelecanoides* is that the `CREx` scenario from *C. kishinouyei* to *B. nectabanus* does not contain the transposition found towards *C. kishinouyei*. This problem could be resolved with a method enumerating all al-

ternative parsimonious TDRL (and transposition) scenarios. Such a method is discussed in Chapter 5.

Some of the rearrangements identified by `TreeREx` in the teleost data set have been reported in the literature while others are reported here for the first time.

Boore (2001) is a rich source of information for gene rearrangement studies and it reports many of the rearrangements `TreeREx` derived from the data set. In Boore (2001) the transposition and the inversion to *C. myriaster*, the transposition to *C. Sloani*, the two transpositions to *M. affine*, as well as the transposition to *C. kishinouyei* are reported in agreement with the results of `TreeREx` for this data set. The gene order of *G. gracile* in Boore (2001) differs from the gene order used in this study, i.e the tRNAs Pro and Glu — which are involved in the two inverse transpositions — are on the other strand. Two transpositions are reported in Boore (2001) instead of the two inverse transpositions. When the data set is changed accordingly, `TreeREx` predicts two transpositions on the edge to *G. gracile* without changing any other predicted event. Thus, also these two rearrangements may be considered to be in agreement with Boore (2001). In Miklós and Hein (2005) an inverse transposition is predicted for the pair of gene orders of *C. kishinouyei* and *G. gracile*. This is compliant to the result returned by `TreeREx`, but is also mislead by the potentially wrong annotated tRNAs.

The gene orders of *E. pelecanoides*, *B. nectabanus*, *A. minima*, *C. bermudensis*, and *A. minima* are not included in Boore (2001). But most of the rearrangements predicted by `TreeREx` are reported in the literature. The difference of *TV* and the gene order of *G. gracile* is explained by two rounds of tandem duplication random loss in Miya and Nishida (1999). While tandem duplication random loss can explain the differences in the order of the genes, it fails to explain the differences in the strandedness of the tRNAs Pro and Glu. But this inconsistency vanishes when the annotation of the tRNAs Pro and Glu is assumed to be wrong (as described above). The transposition and inversion to *C. myriaster* was also reported in Dowton et al. (2003); Inoue et al. (2001). A potential mechanism for the rearrangement found in *C. bermudensis* is discussed in Mabuchi et al. (2004) consistent with the rearrangement reported here. In Inoue et al. (2003) a mechanism of mitochondrial gene rearrangement in gulper eels was proposed, which exactly corresponds to the TDRL as found by `TreeREx` leading towards *E. pelecanoides*. The involved nodes in the phylogenetic tree are all inferred consistently, hence there is a very strong support for this TDRL. Interestingly, another TDRL was found in the mitogenomes of *Teleostei*, namely the TDRL leading towards *B. nectabanus*. This TDRL was first identified by `TreeREx` (Bernt et al., 2008a). The transpositions to *A. minima* was not found in the literature.

The mitogenomes of *Teleostei* can be seen as a relatively easy data set, as many of the leaf nodes have the typical vertebrate gene order as the assigned permutation. Nevertheless,

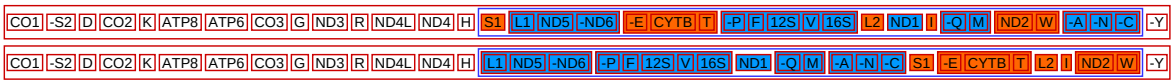


Figure 4.16 – TDRL inferred by algorithm `TreeREx` for the scenario of a typical vertebrate gene order (top) towards the gene order of *Bregmaceros nectabanus* (bottom)

besides the inverse transposition all considered types of rearrangement operations occur. The computation time for `TreeREx` was 0.2 seconds on a laptop with 2.0 GHz processor.

4.5 Exploring the potential of the rearrangement explorer

`CREx`'s potential has hardly been exhausted by the computation of pairwise rearrangement scenarios or the mapping of rearrangements to the edges of a given phylogenetic tree as presented in Sections 4.2 and 4.3. In the following another possible application of `CREx` is suggested and explored to some extend. The results presented here for the first time clearly show additionally that the rearrangements reconstructed by `CREx` agree to a very large extent with the literature.

4.5.1 The approach

The goal of the method presented in the following is mainly the comparison of the `CREx` reconstructions for mitochondrial gene orders with the available literature. Because this comparison is manual it is virtually impossible to consider all pairs of gene orders. Hence, a subset of the pairwise comparison has to be chosen. This is done based of the results gained in the simulation study presented above.

In Section 4.4.1 the relation of `CREx`'s reconstruction quality on the absence of prime nodes in the strong interval trees was shown empirically. More precisely, the absence of prime nodes is an indicator for good quality of the `CREx` reconstruction. Thus, a method restricted on pairwise comparisons with linear strong interval trees may obtain high quality results. For being a useful method, enough remaining pairwise comparisons are important. Fortunately, prime nodes are often absent in biological, i.e. mitochondrial gene order, data. Table 4.3 gives the percentage of pairs of mitochondrial gene orders where no prime node occurs for the improved mitochondrial data set described in Appendix A. This can be seen best for the mitochondrial gene orders of the *Chordates* and their subgroups. More than half of the pairwise comparisons of the mitochondrial gene orders have no prime node in the corresponding strong interval trees. In the *Actinopterygii* even 68.8% of the pairs are prime node free. The *arthropod* gene orders are more diverse, i.e. fewer pairs have only linear nodes, except for the *hexapod* gene orders where more than 50% of the pairwise comparisons have no prime node. For many gene orders in the mitochondrial data sets there exists at least

	act	nac	cho	ehx	cru	hex	cmy	art	abm	np	all
l %	68.8	59.5	60.2	3.6	16.3	51.3	9.4	19.6	9.5	7.8	10.2
L %	89.3	94.9	91.8	36.4	75.0	83.3	59.3	71.4	65.2	66.7	71.9

Table 4.3 – Properties of the strong interval trees for all pairs of unique gene orders in the improved mitochondrial data sets from Appendix A; l: percentage of gene order pairs having no prime node; L: percentage of the gene orders being in at least one pairwise comparison without a prime node

another gene order such that the corresponding strong interval tree has no prime node. For the complete data set, including all unique gene orders, over 70% of the gene orders are part of at least one such pair. For the *Chordata* it is close to or more than 90%. Also for many of the *arthropod* gene orders there exist often a comparison that has a strong interval tree with only linear nodes. Except of the *Myriapoda* and *Chelicerata* more than 70% of the gene orders are included in such a pair.

In many phylogenetic reconstruction methods the data of weakly related species is never — or only indirectly — used. For example, established methods like Fitch's algorithm for parsimonious ancestral character state reconstruction (Fitch, 1971) and the Neighbor-joining algorithm (Saitou and Nei, 1987) either only compare closely related species, i.e. with a small distance in the given phylogenetic tree respectively between the data, or exclude weakly related pairs. `TreeREx` is a further example fur such methods comparing only closely related species, where the relation of species is given by the tree. Thus, it is a valid strategy to omit the comparison of two gene orders if they are not closely related. There are many possible options to decide whether two gene orders are closely related. Here the number of events inferred by `CREx` is used as a measure of dissimilarity. Considering only closely related gene order pairs is in particular useful because the reconstructions of `CREx` are of high quality as indicated for the simulated data sets.

Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a list of unique gene orders. A directed graph $G = (V, E_l \cup E_p)$ is defined such that each node in $V = (v_1, \ldots, v_k)$ corresponds to one of the unique gene orders in Π and the edges represent rearrangement scenarios reconstructed by `CREx`. The edge set is defined as follows. The set E_l is a subset of the edge set representing comparisons based on linear strong interval trees defined as follows. Two nodes $v_i, v_j \in V$, with $1 \leq i \neq j \leq k$, corresponding to two gene orders $\pi_i, \pi_j \in \Pi$ are connected by an edge in E_l if

i) if the strong interval tree for $\{\pi_i, \pi_j\}$ is linear, and
ii) there is no gene order $\pi_h \in \Pi$, with $h \neq i$ and $h \neq j$, such that the strong interval tree for $\{\pi_h, \pi_j\}$ is linear and $d(\pi_i, \pi_h) < d(\pi_i, \pi_j)$. The distance d is given by the length of the corresponding `CREx` scenarios (in the case of alternative scenarios the shorter alternative is regarded).

The graph defined by the edges E_l excludes completely the possibility to analyse TDRLs. Because this kind of rearrangements is of interest, the edge set E_p is considered additionally. There is an edge $(v_i, v_j) \in E_p$ if

i) the rearrangement scenario computed by `CREx` from π_i to π_j includes exactly one TDRL,
ii) v_i and v_j are in different connected components in the graph (V, E_l), and
iii) $d(\pi_i, \pi_j) = \min_{h \in [1:k]}(d(\pi_h, \pi_j))$.

4.5.2 Results

The graph as defined above has been computed for the set of all 185 known unique complete *Metazoan* gene orders of the improved gene order data set presented in Appendix A without *Nematoda* and *Platyhelminthes*. The computation needed 30 seconds on a laptop with a 2.0 GHz CPU. The 185 nodes of the resulting graph are organised in several connected components. Most of the connected components are small: 29 nodes are singletons, nine components contain two nodes, six components contain three nodes, there is a component of size five, and one of size eight. Additionally there are two huge components containing together more than half of the nodes. One has 45 nodes and represents, with the exception of one *Priapulida*, gene orders from *arthropod* species and the other component contains 62 nodes which corresponds to gene orders of *chordate* species, excepting the *Xenoturbellida Xenoturbella bocki* and the *Hemichordata Balanoglossus carnosus*.

In the following connected components of the resulting graph which include more than one node are analysed. Furthermore, other possible improvements are discussed based on the gained insights. Note, the study presented in the following is not intended to be phylogenetically conclusive. Also the massive amount of available literature published in the last decades can not be presented here exhaustively.

Nodes of the graph are drawn as rectangles, where the label gives the GenBank accession number of one representative species sharing the corresponding gene order. The number of species with the same gene order is given in parentheses if greater than one. Next to the node, the Latin name of the representative species is given. Note, the distance of the nodes in the figures does not represent the evolutionary, i.e. rearrangement, distance. Edges are coloured with respect to the rearrangement scenario they represent. Red represents TDRLs, green represents inversions, and blue edges stand for transpositions. For mixed scenarios, the colours are mixed in a weighted fashion, such that the intensity of the three colours represent the number of corresponding rearrangements in the scenarios. For colouring inverse transpositions are treated as inversion plus transposition, i.e. a singe inverse transposition is shown blue mixed with green. Scenarios which are commutative are shown as undirected edges. The direction of scenarios containing TDRLs is indicated by a directed edge. Each edge is labelled with the unique identifiers of the rearrangement that are predicted by `CREx`.

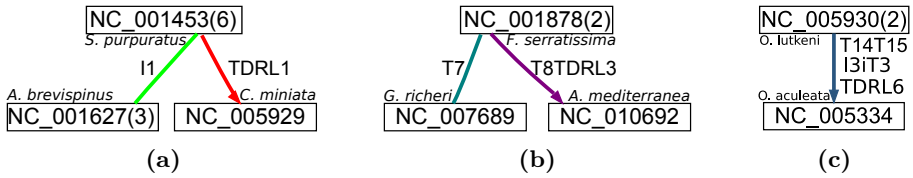


Figure 4.17 – Connected components of size greater than one including gene orders from Echinodermata a) *Echinoidea* (top node), *Asteroidea* (bottom left), and *Holothuroidea* (bottom right node and one in the top node) b) *Crinoidea*, c) *Ophiuroidea*

An index of all identified rearrangements is given in Appendix B. The *Chordata* component is not analysed here. The component is shown in Appendix B.

Echinodermata

The gene orders from the *Echinoderm* species in the data set are organised in three connected components shown in Figure 4.17.

Figure 4.17a shows the three nodes representing the unique gene orders of *Echinoidea*, *Asteroidea*, and *Holothuroidea*. Interestingly, one gene order of a recently sequenced *holothuroid* species (*A. japonicus*) shares the gene order with the known *Echinoids*. This strengthens the *Echinozoa* hypothesis which joins *Echinoidea* and *Holothuroidea* (Hypothesis B1 in Perseke et al. (2008)). Figure 4.17b represents the rearrangements between the unique gene orders of *Crinoid* species. All rearrangements shown in Figures 4.17a and 4.17b, i.e. I1, T7, T8, TDRL1, and TDRL3, are reported in the literature (Perseke et al. (2008) and references therein) and agree with the results presented in Sections 4.4.2 and 4.4.3.

The rearrangement scenario for the *ophiuroid* species (see Figure 4.17c) is discussed in Scouras et al. (2004) but limited to the rearrangements of protein coding genes. The inversion of CYTB, NAD1, and NAD2 reported there is consistent to the inversion I3 that includes in addition the tRNAs D, L2, ND1, I, Q, and N. The transposition of the tRNA-Cluster V, C, and Y to a distant position (T14), exchange of the cluster L1, V, A, C, and Y and the cluster E, P, 12S, and F (T15), the inverse transposition of tRNA-T to a distant position (iT3), as well as TDRL6 do not change the relative position of protein coding genes and are therefore not reported in Scouras et al. (2004). The correctness of rearrangement scenario can not be established on the basis of the given data.

The three *Echinoderm* components are not connected because the rearrangement scenarios within the components are shorter than the rearrangement scenarios between gene orders in different components. For example the rearrangement scenario from the *crinoid* component to the *echinoid, asteroid*, and *holothuroid* component presented in Perseke et al. (2008) and Section 4.4.3 contains three rearrangements. For each gene order in the two components

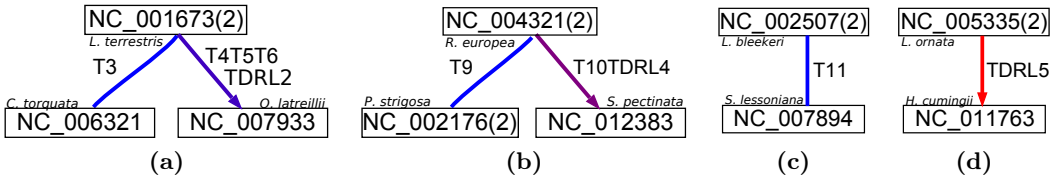


Figure 4.18 – Connected components of size greater than one including gene orders from a) *Annelida* and b-d) *Mollusca*; b) *Gastropoda*; c) *Cephalopoda*; d) *Bivalvia*

there exists another gene order in the same component which can be reached with less than three rearrangements.

Mollusca and Annelida

Figures 4.18 and 4.19 show the connected components including *mollusc* and *annelid* gene orders.

Figure 4.18a shows the only connected component representing gene orders from *annelid* species. It contains four gene orders organised in three nodes. The fifth *annelid* gene order of *P. dumerilii* is a singleton node. Jennings and Halanych (2005) noted that the tRNA-K is at a different position in *C. torquata* indicating that T3, which is a transposition of tRNA-K, is correctly reconstructed. The rearrangement scenario for *L. terrestris* and *O. latreillii* was not explicitly described before, but Bleidorn et al. (2006) noted that "at least five rearrangement events have to be assumed". `CREx` reconstructs for this pair of gene orders transpositions of tRNA-G (T4), tRNA-L1 (T6), and a swap of two blocks containing eleven tRNAs, both rRNAs and three protein coding genes plus a TDRL(2) are predicted. Note that the `CREx` comparison in the other direction yield a minimal four transposition scenario, which is not included in the graph because the strong interval tree has a prime node and currently only scenarios containing exactly one TDRL are accepted by the method.

The *mollusc* gene orders are organised in five connected components of size greater than one. Three *mollusc* gene orders are singletons nodes (*S. lobatum*, *G. eborea*, and *P. dolabrata*).

Figure 4.18d shows two *Bivalvia* gene orders that are separated by a single TDRL(5). This TDRL is not reported in the literature so far, and is another good example for the TDRL model of genome rearrangement because three transpositions are necessary alternatively and in the opposite direction two TDRLs are needed. Moreover, the TDRL includes not only tRNAs but also two protein coding genes.

Rearrangement scenarios for three of the *gastropod* gene orders are shown in Figure 4.18b. The transposition of tRNA-C (T9) to *P. strigosa* was earlier mentioned by Kurabayashi and Ueshima (2000) (in comparison with more distant *Gastropoda*) and Grande et al. (2008). The exchange of tRNAs D and F with CO2 (T10) was indicated in Grande et al.

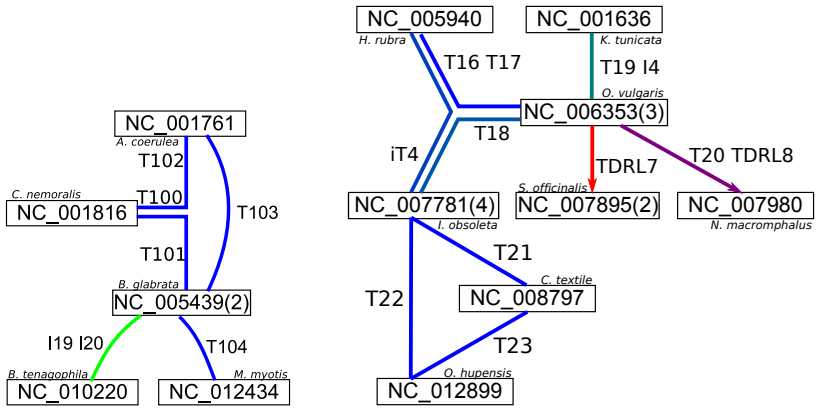


Figure 4.19 – The two larger connected components including mitochondrial gene orders from *mollusc* species; a) five *Gastropoda*; b) four *Gastropoda*, the *Chitonid K. tunicata*, and three *Cephalopoda S. officinalis*, *N. macromphalus*, and *O. vulgaris*

(2008) but the remaining scenario, reconstructed as a TDRL by `CREx`, was not resolved. Note that TDRL4 can be explained alternatively by two transpositions.

Scenarios for five more *gastropod* gene orders are given in the connected component shown in Figure 4.19a. The inversions I19 and I20 of two single tRNAs (L2 and N) are not reported in the literature. The transpositions T100 (swap of the pairs CO3 and T with ND4 and S1), T102 (transposition of tRNA-P and ND6, tRNA-A), T103 (swap of tRNA-A with tRNA-P), T104 (transposition of ND4L with CO2, CYTB, tRNA-D, -F, and -C) are given in Grande et al. (2008). T100 is also consistent with an earlier study that considered only the arrangement of protein coding genes and rRNAs (Hatzoglou et al., 1995). T100 and T102 have also been described in Yamazaki et al. (1997). Transposition T101 (swap of ND6 and tRNA-P) was not found in the literature and suggests an alternative scenario. The presented results suggest three possible scenarios for *C. nemoralis*, *A. coerulea*, and *B. glabrata*:

i) *A. coerulea* is the ancestral arrangement of genes, and the gene orders of *C. nemoralis* and *B. glabrata* are derived via T102 and T100 respectively T103,

ii) *B. glabrata* is the ancestral arrangement of genes, and the gene orders of *C. nemoralis* and *A. coerulea* are derived via T101 and T100 respectively T103,

iii) the gene arrangement that is separated by T100 from *C. nemoralis*, by T101 from *B. glabrata*, and by T102 from *A. coerulea* is ancestral.

The largest connected component of *mollusc* gene orders is given in Figure 4.19b. It contains gene orders from species of different *mollusc* subgroups. It contains five *gastropod*, three *cephalopod* (*O. vulgaris*, *N. macromphalus*, and *S. officinalis*), and one *chitonid* (*K. tunicata*) gene order. The transpositions T21, T22, and T23 that transpose tRNA-L2 are not described

in the literature. Note that T21 may be caused by an annotation error because the position of the tRNA reported in the data set differs from the location reported in BANDYOPADHYAY ET AL. (2008) where the gene order of *C. textile* is given equivalent to the gene order of *I. obsoleta*. Transpositions T16 (transposition of tRNA-D), T17 (transposition of tRNA-N with the pair tRNA-I and ND3), and T18 (swap of the position of tRNA-C and tRNA-Y) between the gene orders of *H. rubra* and *O. vulgaris* are reported in BANDYOPADHYAY ET AL. (2006); MAYNARD ET AL. (2005). All the rearrangements reconstructed by `CREx` between *H. rubra*, *I. obsoleta*, *O. vulgaris*, and *K. tunicata*, i.e. T16, T17, T18, iT4 (inverse transposition of a gene cluster consisting of eight tRNAs, both rRNAs, and six protein coding genes), I4 (tRNA-P), and T19 (swap of tRNA-D and CO2), are reconstructed as reported in BANDYOPADHYAY ET AL. (2006) (except that iT4 is presented as two separate inversions). Note that the `CREx` scenarios between the gene orders of *O. vulgaris*, *H. rubra*, and *I. obsoleta* match perfectly, i.e. they have many common rearrangements as indicated in Figure 4.19b. This immediately proposes the gene order that is separated by iT4 from *I.obsoleta*, by T18 from *O. vulgaris*, and by T16, T17 from *H. rubra* as the ancestral gene arrangement for at least two of the three gene arrangements. Because *I. obsoleta* and *H. rubra* are *Gastropod* species and *O. vulgaris* is a *Cephalopod*, it is likely that the gene order is the ancestral of the two *Gastropod* gene orders and derived from the *O. vulgaris* gene order. Assuming any of the three gene orders as ancestral would not be parsimonious. TDRL7 is presented in AKASAKI ET AL. (2006) (the other TDRL towards the gene order of *W. scintilans* also reported in the study is not included here because it contains duplicate genes). The scenario to *N. macromphalus* is reported differently in BOORE (2006a). Instead of TDRL8 a "transposition of two large blocks and transposition of F" are proposed. The transposition of tRNA-T (T20) is reported equivalently.

The transposition of tRNA-V, -I, -W, 12S, and 16S to a distant position (T11) separating the gene orders of the *Cephalopoda L. bleekeri* and *S. lessoniana* presented in Figure 4.18c is described in AKASAKI ET AL. (2006).

Arthropoda

The gene orders of the *Arthropoda* are clustered in nine connected components. The six components of size two are given in Figure 4.20 and Figure 4.21 shows the two components of size three. Furthermore, there is a huge component containing 45 of the 77 unique *arthropod* gene orders. For an easier representation the large component is presented in two parts. Figure 4.22 includes all nodes which have only one adjacent edge, the nodes which are at the other end of the edge, and the edge itself (plus one line graph). Figure 4.23 contains the remaining of the connected component. The two graphs have only three nodes in common (NC_002010, NC_000844, and NC_002355).

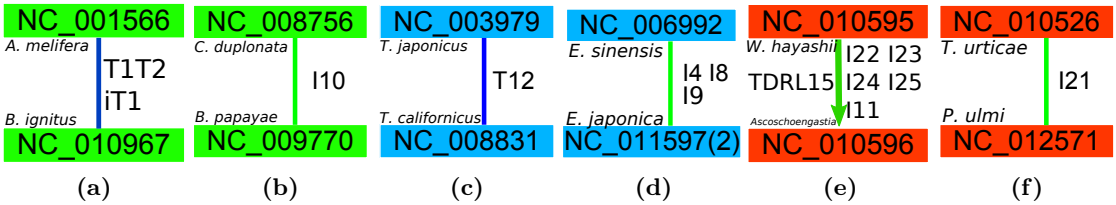


Figure 4.20 – The connected components of size two including mitochondrial gene orders from *Arthropod* species; a,b) *Hexapoda*; c,d) *Crustacea*; e,f) *Chelicerata*

Four gene orders from *Hexapoda* are included in connected components of size two. The gene orders of honey bee and bumble bee represented in Figure 4.20a are separated by the transposition of the pair of genes tRNA-S1 and tRNA-E to a distant position (T1), the swap of the positions of tRNA-T and tRNA-P (T2), and the inverse transposition of tRNA-Q to the other side of tRNA-M (iT1). All three rearrangements are mentioned in Cha et al. (2007). Note the T2 is found several times more in the large *hexapod* component shown in Figure 4.23, this is discussed below. The inversion of S1 (I10) separating *C. duplonata* and *B. papayae* (Figure 4.20b) was not found in the literature, but problems in the strand assignment for *C. duplonata* are reported (Cameron et al., 2006b).

Nine unique *crustacean* gene orders are found in two components of size two (Figures 4.20c and 4.20d) and one component of size three Figure 4.21a. The transposition of tRNA-W to a distant position (T12) reconstructed by `CREx` for the gene orders of *T. japonicus* and *T. californicus* (Figure 4.20c) is not discussed by Burton et al. (2007). This is likely caused by an error in the annotation of tRNA-W, which is located in the 5' end of the 16S rRNA. tRNAscan locates the tRNA at a different position. This problem is not resolved. Three separate inversions differentiate the gene orders of *E. sinensis* and *E. japonica*, i.e. the inversions of tRNA-P (I4), 12S (I8) and 16S (I9) (Figure 4.20d). These inversions are not reported in the literature (Sun et al., 2005). The gene arrangements of the three species

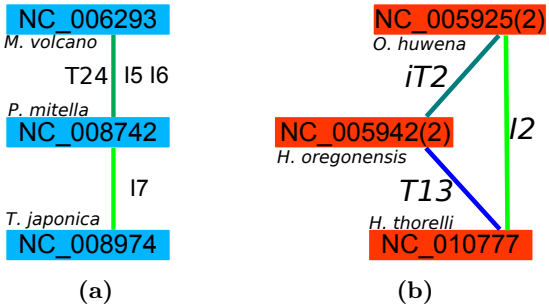


Figure 4.21 – The connected components of size three that include mitochondrial gene orders from *Arthropod* species; a) *Crustacea*; b) *Chelicerata*

in Figure 4.21a are discussed in Lim and Hwang (2006). Unfortunately, the arrangements given in the article differ heavily from the annotations found in the data set. Nevertheless, all rearrangements found by the `CREx` approach, i.e. the transposition of tRNA-K and tRNA-Q to a distant position (T24), the inversion of the set of genes and tRNAs including ND4L, ND4, H, ND5, P, and F (I5), the inversion of tRNA-Y (I6), and the inversion of the pair of tRNAs C and Y (I7) can be recognised. Note that the gene order of *P. polymerus* is not included here because it contains three (adjacent) copies of the tRNA-C. See also Hwang et al. (2001) where tRNA-C and tRNA-Y are noted to be mobile.

Also nine unique gene orders from *Chelicerata* are included in smaller connected components of size two and three (see Figures 4.20e, 4.20f and 4.21b). The inversion of tRNA-F (I21) between *T. urticae* and *P. ulmi* (Figure 4.20f) was not found in the literature. Also the long scenario consisting of five inversions affecting big parts of the gene order (I11, I22-I25) and one TDRL (TDRL15) (Figure 4.20e) is not presented in the literature and deserves more attention. The three rearrangements shown in Figure 4.21b affect the tRNA-I. This is a transposition of tRNA-I (T13), an inversion of tRNA-I (I2), and the combination of both, i.e. an inverse transposition of tRNA-I (iT2). The literature for these rearrangements is incomplete and partially contradictory. Fahrein et al. (2007) reports a transposition of tRNA-I twice: between *O. huwena* and *H. oregonensis* (there rearrangement 12 in Figure 6) and on the edge towards the outgroup of the two species (there rearrangement 5 in Figure 6). This is in contrast to the inverse transposition reported here. Qiu et al. (2005) reports (among other rearrangements) an inversion of tRNA-I in the comparison of the gene arrangements of *O. huwena* and *L. polyphemus*. Masta and Boore (2004) compares the gene orders of *H. oregonensis* and *L. polyphemus* and reports a transposition which is very similar to the transposition of tRNA-I reported here (the same sets of genes are exchanged and tRNA-T is transposed). Thus, one can speculate that the gene order of *H. thorelli* is the ancestral.

In the large connected component containing *Hexapoda* (Figures 4.23 and 4.22) there are two unique gene orders, i.e. nodes, that represent species from different taxonomic groups of interest. The node labelled NC_000844 in the centre of Figure 4.23 represents gene orders of *pancrustacean* species, i.e. 90 *Hexapod* and 14 *Crustacean*. The gene order corresponding to this node is considered to be the ancestral *Pancrustacean* gene order throughout the literature (e.g. Lavrov et al., 2004). The node labelled NC_012421 represents the gene orders of two *Myriapod* and one *Crustacean* species. The gene order of *L. polyphemus* is regarded as ancestral *arthropod* gene order (Boore et al., 1995).

First the rearrangements shown in Figure 4.22 are analysed, starting with the rearrangements leading to *Chelicerata* followed by the rearrangements leading to *Myriapoda* and *Crustacea*. The inverse transposition of tRNA-C (iT28) is described in Black 4th and Roehrdanz (1998); Shao et al. (2004), but instead of TDRL19 two transpositions are proposed. Note, there two different species with the same gene orders are used, i.e. *I. Hexagonus*

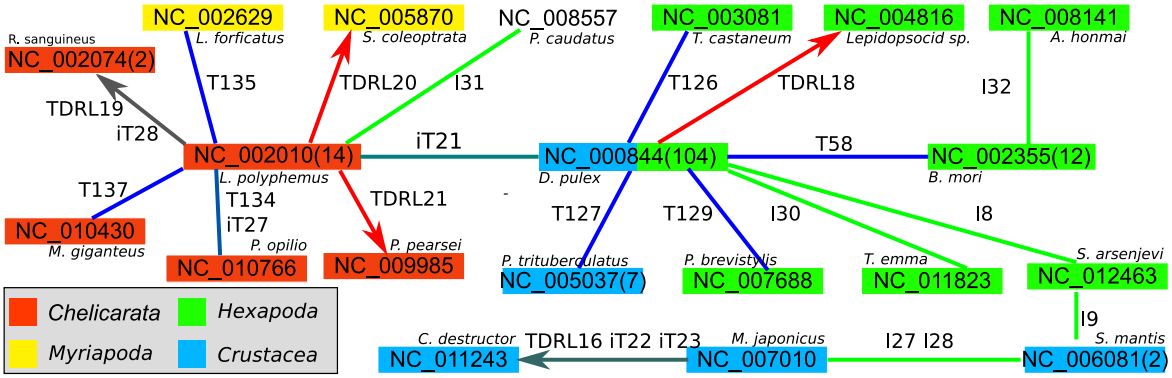


Figure 4.22 – Subgraph of the large connected component that includes mitochondrial gene orders from *Arthropod* species; here the nodes which have only an adjacent edge to the nodes labelled NC_002010, NC_000844, and NC_002355 and a line subgraph starting from NC_000844 are shown together with the edge and the adjacent nodes; the remaining part of the graph is shown in Figure 4.23

and *C. capensis*. The transposition of Q (T134) and the inverse transposition of I (iT27) to the gene order of *P. opilio*, and the exchange of the tRNAs R and A with S1, E, and N (T137) in the gene order of *M. giganteus* were not found in the literature. TDRL21 is a striking example of the tandem duplication random loss model of genome rearrangement. Fahrein et al. (2007) gave a slightly different explanation for differences between the *L. polyphemus* and *P. pearsei* gene orders. This is a TDRL and the transposition of tRNA-N. The reason for this difference is that in the data set used here the control region is not included in the gene orders but in Fahrein et al. (2007) it is. Note that five transposition are necessary as alternative for TDRL21 and the three TDRLs are needed for the rearrangement in the opposite direction. TDRL20 is another convincing example of the tandem duplication random loss model of genome rearrangement which is also presented slightly different in the literature. Seven transpositions are necessary for an alternative explanation of TDRL20 and three TDRLs are needed for the rearrangements in the opposite direction. Note that Negrisolo et al. (2004) postulated “at least 10 translocations". Podsiadlowski et al. (2007) explained the rearrangement as a TDRL and a transposition of tRNA-I. The reason for this difference is again the inclusion of the control region in the comparison. The transposition of tRNA-C to *L. forficatus* (T135) can be found in Hwang et al. (2001) where tRNAs C and Y are reported to be mobile in *Arthropod* gene order evolution. The swap of the positions of tRNA-E and tRNA-F (T126) is another instance where the data set differs from the annotation given in the literature (Friedrich and Muqim, 2003). In Shao et al. (2001) tandem duplication random loss was suggested as the cause of the rearrangement in the undescribed *Lepidopsocrid* species. It was remarked that for nine of the eleven moved genes the TDRL model matches. The actual TDRL (TDRL18) is given here for the first time. The other

species (*T. imaginis*) reported in this study is unfortunately not included here because of duplications. Also the inversion of the three tRNAs S1, E, and N (I30) were not found in the literature. The transposition of tRNA-H (T127) can be found in Yamauchi et al. (2003) and the transposition of tRNA-Y (T129) to *P. brevistylis* is reported in Podsiadlowski (2006). The subsequent inversions of 12S (I8) to *S. arsenjevi*, 16S (I9) to *S. mantis*, and 12S, V, and 16S (I27) and V (I28) to *M. japonicus* are potential misannotations because all three gene orders are reported to be equivalent to the gene order of *D. pulex* in the literature (Cook et al., 2005; Hua et al., 2008; Yamauchi et al., 2004). In Miller et al. (2004) the gene orders of *C. destructor* and *D. pulex* have been compared and different positions for eleven genes have been noticed where for two genes inversion is involved. Furthermore, "for nine of the translocations, the 'duplication/random loss' mechanism" was suggested to be plausible and "a minimum of five independent duplication/random loss events" have been postulated, but "the exact phylogenetic distribution of the *C. destructor* gene order remains yet to be determined". The reconstruction of `CREx`, here in comparison with *M. japonicus*, gives a reconstruction that matches the rough description perfectly. An inverse transposition of tRNA-P (iT22) and tRNA-V (iT23), plus TDRL16 is reconstructed. Lee et al. (2006) report a transposition of tRNA-M (T58) in the gene order of *A. honmai* with respect to the gene order of *D. pulex*, but the inversion of tRNA-H found (I32) by `CREx` was missed. The huge inversion of ten tRNAs, two rRNAs, and six protein coding genes (I31), separating the gene orders of the *Priapulid P. caudatus* and *L. polyphemus*, is described in Webster et al. (2006).

The rearrangements shown in Figure 4.23 are discussed in the following. Starting at the top left continued roughly in anti clockwise direction. In Lavrov et al. (2002) the gene orders of *L. polyphemus* (=*I. hexagonus*) and *N. annularus* have been compared and a tandem duplication non-random loss rearrangement and a transposition of tRNA-T have been proposed. That is in each copy only genes on the same strand are lost (with the exception of tRNA-C). `CREx` reconstructs the same rearrangements (T141 and TDRL22) but with an intermediate step via the gene order of *Nothopuga sp.* by a transposition of tRNA-P (T136) (Fahrein et al., 2007). TDRL22 has to be studied in more detail in the future because of the direction information given by the TDRL because not all *Myriapod* are found "below" the corresponding node. The inverse transposition of tRNA-W (iT29) and transposition of tRNA-T (T144) are reconstructed as in Woo et al. (2007), where it was speculated that the transposition is derived from the pre-"non random loss" tandem duplicated gene arrangement that gave rise to the *N. annularus* gene order. The reconstructed rearrangements between *I. hexagonus* and *S. causeyae* are nearly as in Podsiadlowski et al. (2007). The swap of tRNA-M and tRNA-Q (T51), and tRNA-T and tRNA-P (T2) are equivalently represented; the transposition of tRNA-V (T105) is presented as TDRL because of additional evidence from sequence data, and the inversion of tRNA-N (I20) is not given and probably the result of

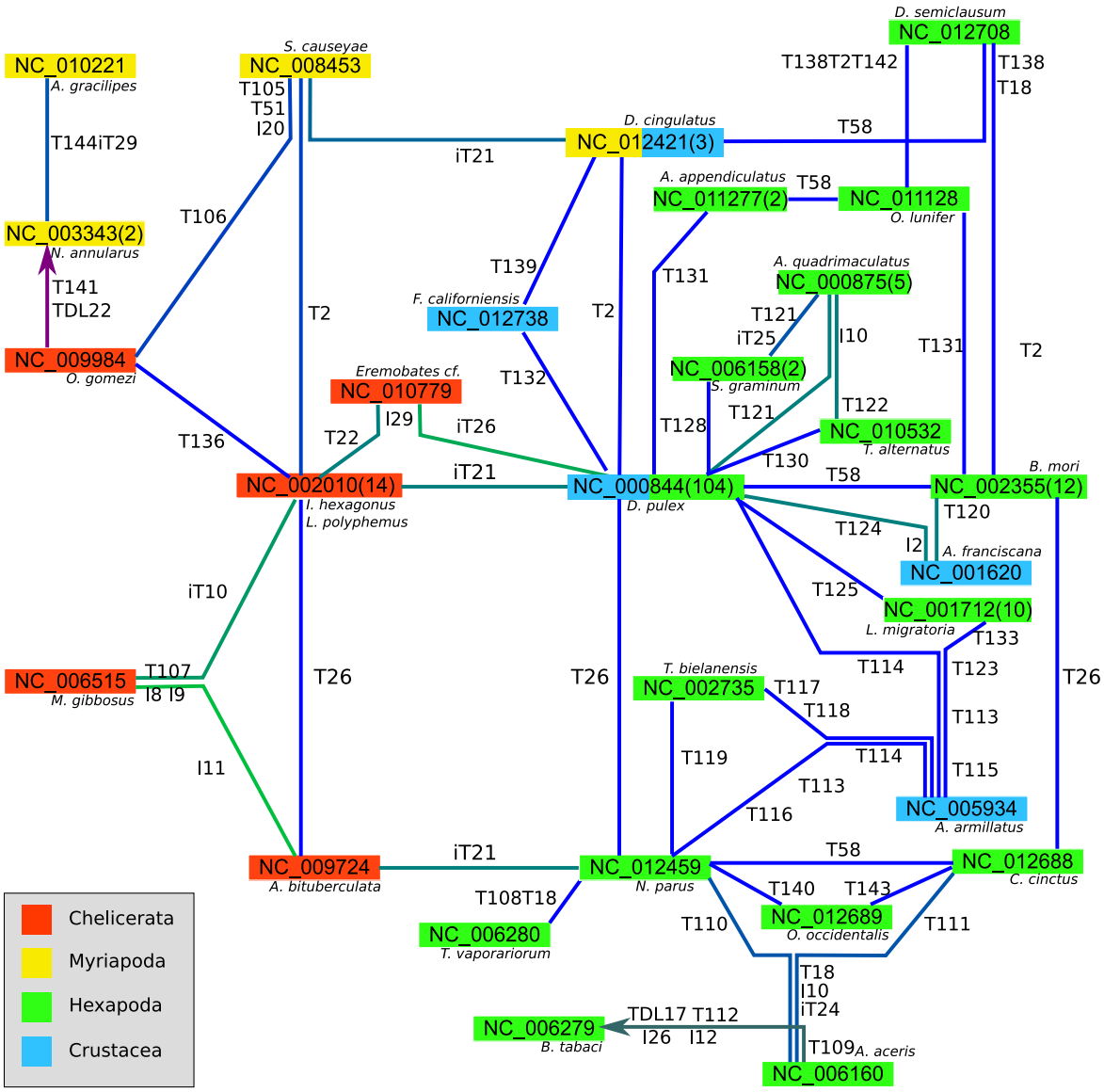


Figure 4.23 – The remaining subgraph of the large connected component that includes mitochondrial gene orders from *Arthropod* species; here all nodes and edges, not shown in Figure 4.22, are included

another annotation error in the data set. The transposition T106 of tRNA-P is an alternative scenario from *O. gomezi* was not found in the literature. The gene arrangement shared by *D. cingulatus*, *E. carinicauda*, *P. gutta* is analysed in Hua et al. (2008); Shen et al. (2009) and the swap of tRNA-T and tRNA-P (T2) is mentioned in both studies. The alternative scenarios to *S. causeyae* and *F. californensis* are given here for the first time (*D. semiclausum* was compared with *D.pulex* in Wei et al. (2009) see below).

A rearrangement scenario from *L. polyphemus* to *M. gibbosus* is presented in Dávila et al. (2005), it agrees with the `CREx` scenario in the transposition of tRNA-D (T107) and the inverse transposition of tRNA-Q (iT10). The single inversions of 16S (I9) and 12S (I8) are probably annotation errors. The alternative scenario from *A. bituberculata*, consisting of the inversion of tRNA-Q (I11) and also I8, I9, and T107, was not found in the literature. One can speculate that the scenario from *A. bituberculata* is more likely because I11 is reconstructed twice within *Chelicerata* (see Figure 4.20e), iT10 is very similar to T26 (same two swapped tRNAs), and an inversion is potentially a "simpler" rearrangement than an inverse transposition. The gene arrangement of *A. bituberculata* is discussed in Park et al. (2007). The swap of tRNA-I and tRNA-Q (T26) is reported. It was observed that similar to the *L. polyphemus* gene arrangement the inverse transposition of L2 (iT21) has not happened. Furthermore, the gene arrangement found at *A. bituberculata* was proposed as the *Arthropod* "ground pattern". A rearrangement scenario for *Eremobates cf.* was not found in the literature. The `CREx` scenario swap of L1 and L2 (T22) and inversion of S2 (I29) is a candidate for misannotation that has to be checked. The inverse transposition of L2 (iT26) is similar to iT21, but L1 is not affected because of T22.

For *F. californiensis* no rearrangement scenario was found, too. `CREx` suggests two different transpositions of tRNA-T from the gene order of *D. pulex* (T132) respectively *D. cingulatus* (T139). Which of the two actually happened in the gene order evolution, can not be decided given the available data. Note that also T2 is a transposition of tRNA-T (swap of T and P). Valverde et al. (1994) noted that the only differences between the gene orders of *A. franciscana* and *D. pulex* "are two tRNA genes—the tRNA Q that is located in a different position and the tRNA I that is in a different location and orientation". `CREx` reconstructs two rearrangements that match this description, i.e. the transposition of the pair of tRNAs I and Q (T124) and the inversion of tRNA-I (I2). The transposition of tRNA-I and -Q is also noted in Boore (1999) (see also Flook et al. (1995)). Note that `CREx` reconstructs an alternative scenario to the gene order of *B. mori* consisting of a different transposition of tRNA-I and Q (T120) and also I2. The genome rearrangement reconstructed by `CREx` consisting of the four transposition of tRNA-K (T114), tRNA-Q (T123), tRNA-S2 (T113), and tRNA-L1 (T115) from *D. pulex* to *A. armillatus* is discussed in Lavrov et al. (2004). Interestingly, `CREx` finds three alternative scenarios to *L. migratoria*, *T. bielanensis*, and *N. parus*, each consisting of four transpositions. The transposition of L1 (T115) is common to

all rearrangement scenarios, the other rearrangements are only common to a subset of the alternative scenarios. These scenarios have not been reported before and await a further examination.

The gene arrangement found in *O. occidentalis* (respectively *C. cinctus*) is explained by swap of tRNA-Q and tRNA-I (T26) plus a transposition of tRNA-M (T140) (respectively T58) in Dowton et al. (2009) consistent with the `CREx` reconstruction. The alternative transposition of tRNA-M (T143) to *C. cinctus* was not found in the literature. Also the alternative scenario T26 from *B. mori* to *C. cintus* was not found.

The gene arrangement of the mitochondrial genomes of the whiteflies *A. aceris*, *B. tabaci*, and *T. vaporariorum* is compared to the arrangement found in the *Psyllid P. venusta*(= *D. pulex*) and the *Aphid S. graminum* in Thao et al. (2004). The transposition of S2 (T128) to *S. graminum* was not presented in Thao et al. (2004) because S2 could not be identified, but the tRNA is now included in the data set. The transposition of tRNA-G (T108) and the swap of tRNA-C and tRNA-Y (T18) between the gene orders of *N. parus* and *T. vaporariorum* are reported in Thao et al. (2004) in the comparison between *D. pulex* and *T. vaporariorum* (the additional transposition of tRNA-I and tRNA-Q (T26) is included there). The suggestion, made in Thao et al. (2004), to assume that the gene arrangement, differing from the *D. pulex* gene arrangement by T26 and T18, is ancestral is consistent with the results presented here. That is T26 and T18 are found on the paths from *D. pulex* to all the whitefly gene orders included in the study presented here. But note, T18 is also found to the gene order of *D. semiclausum* which is not a whitefly and in the gastropod gene orders (see Figure 4.19b). The rearrangements to *A. aceris* and *B. tabaci* from the whitefly ancestor presented in Thao et al. (2004) are mostly very similar to the rearrangements reconstructed here or equal. A transposition of tRNA-Q can be found, but tRNA-I was not included in *A. aceris* in Thao et al. (2004). Thus, the addition of tRNA-I, as reported in the data set, matches transposition T110 (transposition of tRNA-I and -Q). Note that the alternative transposition of tRNA-M (T111) from *C. cintus* was not found in the literature. Furthermore, two inverse transpositions are reported in Thao et al. (2004): i) the inverse transposition of CO3, ND3, and the tRNAs G, A, R and ii) the inverse transposition of N (which is adjacent to the genes affected by the other inverse transposition in both gene orders). Contrary to that `CREx` reports an inverse transposition of the combined set of genes, i.e. CO3, ND3, and the tRNAs G, A, R, N (iT24) and a separate transposition of tRNA-N (T109). The inversion of tRNA-S1 (I10) is the same in the `CREx` reconstruction and in Thao et al. (2004). In Thao et al. (2004) different inverse transpositions are assumed for the different whitefly gene arrangements. The `CREx` reconstruction presents an alternative that requires only one inverse transposition. One argument for the `CREx` reconstruction is that the scenario for *B. tabaci* and the whitefly ancestor contains an inverse transposition affecting the same set of genes as iT24 (but to a different position). Instead of this another transposition of the genes

CO3, ND3, and the tRNAs R, G, I, N, and A (T112) is predicted. Instead of the separate inversions of the adjacent tRNAs S1 and E only the inversion of E (I10) is reconstructed. The inversion of S1 (I10) is already found in *A. Aceris*. The transposition of tRNA-Q and inverse transposition of tRNA-D are reconstructed by `CREx` as inversion of tRNA-D (I26) and TDRL17.

Nardi et al. (2001) discuss the arrangement of *T. bielanensis* and propose the swap of positions of tRNA-I and tRNA-Q (T26) and a transposition of tRNA-S2 (T119) with respect to the gene order of *D. melanogaster* (equal to *D. pulex*). The same rearrangement is reconstructed by `CREx`, but with the intermediate step via the gene arrangement of *N. parus*.

The swap of the positions of tRNA-D and tRNA-K (T125) in the gene order of *L. migratoria* with respect to the *D. pulex* gene order is described in Fenn et al. (2008); Flook et al. (1995); Ma et al. (2009). According to Cameron et al. (2006a) this rearrangement is also found in *A. melifera*.

The rearrangement scenario from the gene order of *D. pulex* to the gene order of *A. quadrimaculatus*, consisting of the swap of tRNA-R with tRNA-A (T121) and the inversion of tRNA-S1 (I10), is discussed in Beard et al. (1993). The two alternative scenarios via the gene orders of *S. graminum* respectively *T. alternatus* have not been found in the literature (note that *S. graminum* is discussed above). The alternative rearrangement scenario via *T. alternatus* consists of two separate transpositions of tRNA-A within the tRNA cluster consisting of the tRNAs A, R, N, S, E, F (T122 and T130). The alternative scenario via *S. graminum* also includes the swap of tRNA-R and tRNA-A. Additionally, tRNA-S1 is transposed to a distant location to *S. graminum* and then inverse transposed to its "former" position to *A. quadrimaculatus*.

The swap of tRNA-W and tRNA-C (T131) to *A. appendiculatus* from the gene order of *D. pulex* is noted in Beckenbach and Stewart (2009). The transposition tRNA-M to the other side of the tRNAs I and Q (T58) in the *O. lunifer* gene order is reported in Salvato et al. (2008). The swap of tRNA-W and tRNA-C with respect to the *B. Mori* gene order is most likely caused by a misannotation in the start position of tRNA-C. Thus, the *O. lunifer* gene order has to be regarded identical to *B. mori* gene order.

In Wei et al. (2009) the gene orders of *D. semiclausum* and *D. pulex* have been compared and a transpositions of L2 (T138), and M (with IQ) (T58), as well as the swap of tRNA-C with tRNA-Y (T18) and tRNA-P with tRNA-T (T2) are reported. `CREx` reconstructs all these rearrangement events and gives more information. It is unlikely that the gene order of *D. semiclausum* is directly derived from *D. pulex*. Instead `CREx` proposes that the gene order is derived from either *B. mori* or *D. cingulatus* and because the `CREx` scenarios contain T18 and T138 an ancestral state can be suggested. But note, T18 can be found two more times in Figure 4.23 (*T. vaporariorum* and *A. aceris* respectively *B. tabaci*). The rearrangement

	#	known	diff	new
I	44 (26)	13 (9)	14 (8)	17 (15)
T	109 (71)	70 (42)	14 (11)	25 (24)
iT	18 (14)	10 (8)	4 (2)	4 (4)
TDRL	16 (16)	7 (7)	3 (3)	6 (6)

Table 4.4 – Number of rearrangements found in the analysed connected components of the graph; (#): total number of rearrangements in the graph; (known) number of rearrangements in agreement with the literature; (diff); number of rearrangements reported differently in the literature or which are caused be annotation errors; (new): number of rearrangements that have not been found in the literature; in parentheses the number of unique rearrangements is given; note, the unique numbers for inversions and transpositions do not sum to the total number of unique rearrangements

scenario reconstructed for the comparison with *O. lunifer* is not discussed because of the potential misannotation described above.

Table 4.4 gives statistics of the numbers of rearrangements found in the analysed connected components, i.e. excepting the *Chordata*. Clearly, most of the rearrangements identified by `CREx`, which passed the filter of the new method presented here, are correct, i.e. in agreement with the literature. The rearrangements which are reported differently in the literature or which could not be found in the literature have to be examined in more detail in order to confirm or reject them.

Within the components that have been presented here an unexpected high number of transpositions has been found. This is in disagreement with previous studies. Blanchette et al. (1996) suggested that "transpositions are observed much less frequently than inversions in many evolutionary contexts, and hence should cost much more" but noted that "there are as yet no systematic studies, either within or across major phylogenetic groups, of the relative frequencies of inversion, transposition, and other rearrangement processes, based on any type of comparative mapping". Also Yancopoulos et al. (2005) noted that "large-scale transpositions are much less frequently observed than inversions and translocations". Throughout the literature weighting schemes always put more weight on transpositions (e.g. Bader et al., 2008; Blanchette et al., 1996; Eriksen, 2003, 2002). This is done in order to get no bias favouring transpositions. The results presented here indicate that it is necessary to re-examine the weighting schemes at least for intra phylum comparisons of mitochondrial gene arrangements. Also Dowton et al. (2009) identified 43 transpositions within the presented 67 comparisons of mitochondrial gene orders of *Hymenoptera*. For two bacterial genomes also a high number of transpositions was reported in Dalevi et al. (2002).

For future investigations the rearrangements

i) T26 (swap of tRNA-I and tRNA-Q) (Hua et al., 2008; Nardi et al., 2001; Park et al., 2007)),

ii) iT21 (inverse transposition of L2) (e.g. Boore, 1999; Boore et al., 1998),

iii) T58 (transposition of tRNA-M and the pair tRNA-I, tRNA-Q) (Taylor et al., 1993), and

iv) T2 (swap of tRNA-P and tRNA-T) (e.g. Cha et al., 2007; Hua et al., 2008; Podsiadlowski et al., 2007; Shen et al., 2009))

are of great interest. This is because they are found several times between or within different taxonomic groups and may be examples of convergent or synapomorphic rearrangements, as discussed recently in Dowton et al. (2009). The results presented here will be of great help.

4.5.3 Discussion

Gene arrangement data is usually analysed manually by biologists. Such manual analyses are valuable and indispensable. Nevertheless, manual analysis suffers from many problems. Most importantly the handling of the huge amount of available data is at least tedious and may also be considered as impossible.

Therefore, usually only a very small number of gene arrangements is compared (with notable exceptions (Dowton et al., 2009; Fahrein et al., 2007)), only a subset of the genes is evaluated (usually tRNAs are excluded), or only a part of the arrangements is compared (usually all species are compared to a putative ancestral gene order (Dowton et al., 2009), or based on a phylogenetic tree (Fahrein et al., 2007)). In this way the phylogenetic signal which is or is not contained in gene arrangement data can not be properly analysed or important alternative rearrangement scenarios may be missed.

Already in this basic version the presented method facilitates the analysis of gene arrangements and the reconstruction of the rearrangements. Within less than a minute the results for the complete mitochondrial data set can be computed. It was shown here that `CREx` allows for a comprehensive analysis of the rearrangements, within the connected components, solving many of the problems mentioned above, in an unprecedented and efficient way.

Some simple modifications will further improve the method and permit new analyses. Most importantly the intersection of scenarios, as presented in Section 4.3, will allow for the automatic inference of ancestral states. Methods for finding the rearrangements in between the components of the graph have to be developed. Spanning trees of the connected components may be used for finding phylogenetic trees. This may be especially useful when the edges, i.e. rearrangements, in the graph are considered in weighted fashion. The relation of the

weight of the spanning trees of the weighted graph to the weights of the rearrangements will be of special interest. Also of importance is an extension of the CREx method to handle gene arrangements with unequal gene content.

4.6 Conclusion

In this chapter methods for the reconstruction of phylogeny using the four rearrangement operations relevant for mitochondrial gene orders have been presented, i.e. inversions, transpositions, inverse transpositions, and tandem duplication random loss. The common interval rearrangement explorer CREx computes heuristically a rearrangement scenarios for two given unichromosomal gene orders. The rearrangements are identified by CREx as patterns in the strong interval tree. Methods have been presented to handle alternative scenarios, ordered scenarios, and combinations of inversion and tandem duplication loss events.

The tree rearrangement explorer TreeREx automatically infers ancestral permutations and genomic rearrangement operations in a given phylogenetic tree. The rearrangements on the edges are derived from intersections of pairwise rearrangement scenarios (computed with CREx). To derive plausible reconstruction the notion of consistency and k-consistency of rearrangements with the given phylogenetic tree was introduced.

The algorithms CREx and TreeREx have been analysed empirically on simulated and mitochondrial data sets. Parameters for the simulation of random rearrangement scenarios and properties of the strong interval tree have been identified where CREx gives high quality reconstructions. TreeREx was applied to biological data sets of mitochondrial gene orders of *Echinodermata* and of *Teleostei*. In both data sets the reconstructed genome rearrangement operations are in strong correspondence with published results.

The possibilities of the CREx method have been explored with a simple method. This method selects a reliable subset of the pairwise rearrangements. The decision criterion has been based on the insights gained in the empirical study presented in this work. The method was successfully applied to the complete mitochondrial gene order data set. The comparison with the available literature confirms the reliability of the CREx method for the selected subset of rearrangements.

5 Finding all sorting tandem duplication random loss operations

5.1 Introduction

The study of combinatorial properties of rearrangement operations has gained lots of attention. Of interest are for example distance, sorting, and median problems for the different types of rearrangement operations and combinations of the different types (see Sections 2.3 and 2.5). The most often studied rearrangement operations are inversions and transpositions (Bafna and Pevzner, 1995; Hannenhalli and Pevzner, 1995b). The results on the combinatorics of the rearrangement operations have been applied to infer the phylogenetic relationship of gene orders representing species.

As pointed out earlier in this work, mitochondrial gene orders are a fruitful source for the reconstruction of phylogeny from gene arrangements and rearrangements. In mitochondrial gene orders the so called tandem duplication random loss (TDRL) operation can be found several times in the mitochondrial gene order evolution, e.g. in millipedes (Lavrov et al., 2002) and eels (Inoue et al., 2003). More examples have been presented in Chapter 4. Some authors even considered TDRLs as "being the most important rearrangement operation in vertebrate" mitochondrial genomes (Boore, 2000; Inoue et al., 2003; Mauro et al., 2006). Although some studies use rearrangement models which are based on inversions, translocations, chromosome fissions and chromosome fusions, that also try to include gene duplications and losses (e.g. El-Mabrouk, 2000a; Swenson et al., 2006; Tang et al., 2004), the properties of TDRLs have only rarely been investigated.

The TDRL operation has been studied formally for the first time in Chaudhuri et al. (2006) where a solution for the distance and sorting problem have been presented for the TDRL rearrangement operation. A radix sort inspired algorithm to compute a sequence of TDRLs between gene orders that realises the minimum distance has been presented. Furthermore, the asymmetry of TDRLs and the corresponding distance measure was pointed and the TDRL median problem, i.e. finding a gene order which has a minimum TDRL distance to two input gene orders, was studied. For an introduction to these approaches see Chapter 2.

For phylogenetic inference it is important to identify sorting genomic rearrangement operations, i.e. operations reducing the distance towards a given gene order, when applied to another gene order. One reason is that rearrangement scenarios for two gene orders with a minimum number of operations satisfy the maximum parsimony principle and might therefore be considered as more likely than non sorting operations. Hence, in order to find a realistic scenario for describing the rearrangement relation between two gene orders it is helpful to know all possible sorting operations. In this case it is possible to select a sorting operation satisfying certain properties, e.g. smallest number of involved genes. Also for testing hypotheses on rearrangement models all sorting rearrangements have to be known (Ajana et al., 2002). Further, the use of all equally good solutions in algorithms can have a positive effect on the solution quality, as already shown in the context of genome rearrangements (Bernt et al., 2007b). The problem to enumerate all sorting operation has been investigated in the context of inversions (Ajana et al., 2002; Bernt et al., 2006a; Braga et al., 2008; Siepel, 2003) and recently for the DCJ operation (Braga and Stoye, 2009; Ouangraoua and Bergeron, 2009).

A restricted set of TDRLs is introduced in Section 5.4. In the following methods for the enumeration of all sorting TDRLs are presented and closed formulas to compute the number of sorting TDRLs are derived. In Sections 5.5 and 5.6 two different alternative approaches for these problems are presented for the case of restricted and general TDRLs. The second approach, which is presented here for the first time, has nice combinatorial properties and give a missing combinatorial explanation of previously known results. For the example of mitochondrial gene order analysis the relevance of the presented results is shown.

5.2 Basic definitions

A *permutation of size* n is a permutation of the elements in $\{1, 2, \ldots, n\}$. The element of π at the i-th position is denoted by $\pi(i)$. Note, in this chapter only unsigned permutations are considered. The *inverse permutation* π^{-1} of a permutation π is defined such that $\pi^{-1}(e)$ is the index of element e in π, i.e. $\pi^{-1}(e) = i$ iff $\pi(i) = e$. An *interval* of a permutation π is a set of consecutive elements of the permutation π.

A *binary string* t of length n is a string over an alphabet Σ of size two. The element of t at the i-th position is denoted by $t(i)$. A pair of consecutive elements $(t(i-1), t(i))$ of a string t of length n, with $i \in [2:n]$, is called *transition* at position i iff $t(i-1) \neq t(i)$, the transition is called *xy-transition* with $x, y \in \Sigma$ iff $t(i-1) = x$ and $t(i) = y$. Clearly, a binary string of length n can be defined by specifying the character at the first position and for each pair of consecutive positions if a transition takes place or not. More formally, let (T, E) be a bipartition of $\{1, \ldots, n\}$, so the binary string $t = t(1) \ldots t(n)$ is recursively defined by $t(i-1) \neq t(i)$ iff $i \in T$ (respectively $t(i-1) = t(i)$ iff $i \in E$) and $t(0)$ is defined as one of the

two elements in Σ. For example in the following $t(0) = 2$ will be used for binary strings over $\Sigma = \{1, 2\}$. Note, $t(0)$ is only used as base case for the recursive definition of t and is not part of the string t. The set T (respectively E) specifies the pairs of consecutive positions of t where a transition (respectively no transition) takes place.

Example 5.2.1. *The binary string* $s = 2\ 2\ 1\ 2\ 1\ 1$ *of length* 6 *is defined by specifying the elements for each position, i.e.* $s(1) = 2, s(2) = 2, \ldots, s(6) = 1$. *The string* s *has two* 21*-transitions at positions* 3 *and* 5*; and one* 12*-transition at position* 4. *Alternatively,* s *is defined recursively by the set* $T = \{3, 4, 5\}$ *with* $s(0) = 2$. *The construction is:* $s(2) = s(1) = s(0) = 2$ *because neither* 2 *nor* 1 *are in* T*;* $(s(2), s(3))$ *is a transition because* $3 \in T$ *and therefore* $s(3) = 1$*; also* $(s(3), s(4))$ *is a transition because* $4 \in T$*; and so on. Equivalently the string can be recursively defined by the complementary set* $E = \{1, 2, 6\}$.

A *tandem duplication* duplicates an interval of a permutation such that the two copies of the interval appear consecutively without changing the order of the elements in any copy of the interval. The left copy of the interval is called *first copy* and the right copy of the interval is referred to as *second copy*. The terms "first" and "second" must not be interpreted with respect to which is the original and which is the copy. The result of a tandem duplication is not a permutation because it contains duplicate elements. A *tandem duplication random loss* (TDRL) rearrangement τ transforms a permutation (genome) by a tandem duplication of an interval of the permutation and subsequent random loss of one of the copies of the duplicated elements (genes). A TDRL is regarded as an atomic operation, i.e. the tandem duplication and the random loss of the copied genes are not separable. Thus, the result of the application of a TDRL to a permutation is again a permutation. Formally, a TDRL applied to a permutation π is defined as $\tau = (F, S)$, where F specifies the set of elements which are kept in the first copy and S defines the set of elements kept in the second copy. The following conditions must hold for a TDRL $\tau = (F, S)$ of a permutation of length n:

i) $F \subseteq \{1, \ldots, n\}$ and $S \subseteq \{1, \ldots, n\}$,
ii) $F \cup S$ is an interval in π, and
iii) $F \cap S = \emptyset$.

If the context is clear simply τ is used. To identify if an element of a permutation is kept in the first or in the second copy of a TDRL τ, a function $C_\tau : \{1, \ldots, n\} \mapsto \{1, 2\}$ with $C_\tau(e) = 1$ if $e \in F$ and $C_\tau(e) = 2$ for $e \in S$ is defined.

In this chapter a unit cost model for TDRL is assumed, i.e. each TDRL operation has the same weight. This corresponds to the case $\alpha = 1$ in the α^l cost model presented in Chaudhuri et al. (2006) where l is the length of the duplicated interval and $\alpha \geq 1$ some constant parameter. In the assumed cost model a TDRL duplicating only a subset of the genes can be represented by a TDRL duplicating all genes without changing the cost. Formally, let $\tau = (F, S)$ be a TDRL with $F \cup S \subset \{1, \ldots, n\}$ that is applied to a permutation π of size

n. Furthermore, let L (respectively R) be the sets of elements in π to the left (right) of the elements in $F \cup S$. Then the TDRL $\tau' = (F \cup L, S \cup R)$ duplicates the complete permutation and is equivalent to τ, i.e. $\pi \circ \tau = \pi \circ \tau'$. Thus, without loss of generality $F \cup S = \{1, \ldots, n\}$ is assumed throughout this chapter, i.e. the pair (F, S) is a bipartition of the set of elements of π.

Example 5.2.2. *Let for example* $\pi = (3\ 7\ 1\ 5\ 8\ 2\ 6\ 4)$ *be a permutation and let* $\tau = (\{2,5\},\{1,8\})$ *be a TDRL applied to* π. *First, the interval that contains the set* $\{2,5\} \cup \{1,8\}$ *is duplicated in tandem resulting in* $(3\ 7\ \underline{1\ 5\ 8\ 2}\ \underline{1\ 5\ 8\ 2}\ 6\ 4)$ *which is clearly not a permutation. For the purpose of illustration the elements of the first copy are underlined and the elements of the second copy are wavily underlined. This is followed by the loss of one of the copies of each gene specified indirectly by the sets* F *and* S *of the TDRL. That is the stroked out elements are deleted:* $(3\ 7\ \not{1}\ 5\ \not{8}\ 2\ 1\ \not{5}\ 8\ \not{2}\ 6\ 4)$ *resulting in the permutation* $(3\ 7\ 5\ 2\ 1\ 8\ 6\ 4)$. *Note that the TDRL* $\tau' = (\{2,3,5,7\},\{1,4,6,8\})$ *results in the same permutation when applied to* π. *For* τ' *it holds that* $C_\tau(2) = C_\tau(3) = C_\tau(5) = C_\tau(7) = 1$ *and* $C_\tau(1) = C_\tau(4) = C_\tau(6) = C_\tau(8) = 2$.

The sorting and distance problems for the TDRL model of genome rearrangement are defined formally as follows. Let π and σ be two permutations of the same length. The problem of *sorting by TDRLs* is to find a shortest sequence of TDRLs $\tau_1, \ldots, \tau_{d(\pi,\sigma)}$ such that $\pi \circ \tau_1 \circ \ldots \circ \tau_{d(\pi,\sigma)} = \sigma$. The length of the sequence, i.e. the minimum number of TDRLs necessary to transform π into σ, is called the *TDRL distance*, denoted by $d(\pi, \sigma)$.

Without loss of generality the sorting problem is typically — and also in the following — considered for σ being the identity permutation denoted by ι (an exception is Chaudhuri et al. (2006) where $\pi = \iota$ was used). Remark, this assumption is no restriction because by renaming the elements of σ so that it becomes the identity (i.e. $\sigma \circ \sigma^{-1} = \iota$, where $\circ$ denotes the composition of permutations) and applying the same renaming of the elements to π (i.e. $\pi \circ \sigma^{-1}$). The assumed case can easily be generated from every instance.

For a given pair of permutations (π, σ) a sorting TDRL is a TDRL for which for which the distance from π to σ is decreased after applying τ to π, i.e. $d(\pi \circ \tau, \sigma) < d(\pi, \sigma)$. Let $B(n)$ denote the set of all bipartitions of $\{1, 2, \ldots, n\}$ and π, σ be two permutations of size n. The set $\mathcal{S}(\pi, \sigma) = \{\tau | d(\pi \circ \tau, \sigma) < d(\pi, \sigma), \tau \in B(n)\}$ of *all sorting* tandem duplication random loss rearrangements is the set of all TDRLs for which the distance from π towards σ is decreased. If the second permutation is the identity permutation the simplified notation $d(\pi)$ is used instead of $d(\pi, \iota)$ and $\mathcal{S}(\pi)$ is used instead of $\mathcal{S}(\pi, \iota)$.

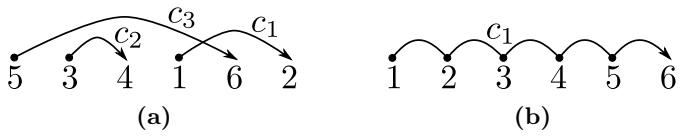


Figure 5.1 – a) The 3 chains of permutation $\pi = (5\ 3\ 4\ 1\ 6\ 2)$, for a detailed description see Example 5.3.1; b) chain for the identity permutation

5.3 TDRLs and chains

In this section the notion of chains of elements in a permutation is introduced and basic properties of chains and TDRLs are stated. Chains serve as the key ingredient to compute all sorting TDRLs in the next sections.

5.3.1 Definition and computation of chains

The chains of a permutation are formally defined as follows.

Definition 5.3.1. *A* chain *of a permutation π is a list $(e_1, \ldots, e_k)$ of elements of π with maximal length, where either $k = 1$ or $e_i - 1 = e_{i-1}$ and $\pi^{-1}(e_{i-1}) < \pi^{-1}(e_i)$ holds for all $i \in [2 : k]$, $k > 1$.*

A chain *connect*s an element $e-1$ with element e iff e is located to the right of $e-1$ in π. Obviously, each element of a permutation belongs to exactly one chain. An example of how a permutation is divided into its chains is given in Example 5.3.1. A *chain is included* in a set of elements (e.g. in a set F of elements kept in the first copy of a TDRL) if all elements of the chain are included in this set of elements.

Let $\rho(\pi)$ be the *number of chains* of a permutation π. An *indexing scheme* for the chains of a permutation can be defined in a straightforward manner: Let c and c' be two chains of a permutation π. A strict total order for chains is defined as follows: $c < c'$ iff $\forall e \in c, \forall e' \in c' : e < e'$. Let $c_1 < \ldots < c_{\rho(\pi)}$ be the total order of all chains of permutation π. Then c_i is said to be the *i-th chain* of π. Furthermore, with the function $C_\pi : \{1, \ldots, n\} \mapsto \{1, \ldots, \rho(\pi)\}$ the index of the chain including a given element in a permutation π is determined, i.e. $C_\pi(e) = i$ iff e is an element of chain c_i.

Example 5.3.1. *The permutation $\pi = (5\ 3\ 4\ 1\ 6\ 2)$ has three chains $c_1 = (1, 2)$, $c_2 = (3, 4)$, and $c_3 = (5, 6)$ (see Figure 5.1a), i.e. $\rho(\pi) = 3$. For example, $(1, 2)$ is a chain because*
a) the element 1 is the start of the chain because it has no predecessor,
b) $\pi^{-1}(1) < \pi^{-1}(2)$ (1 is left of 2 in π), and
c) element 3 is not in this chain because $\pi^{-1}(2) > \pi^{-1}(3)$ (3 is left of 2 in π).

The chain $(1, 2)$ *is the first chain because the other chains only contain elements larger than* 1 *and* 2*, i.e.* $c_1 = (1, 2)$*. It holds that* $C_\pi(1) = C_\pi(2) = 1$*,* $C_\pi(3) = C_\pi(4) = 2$*, and* $C_\pi(5) = C_\pi(6) = 3$*.*

Algorithm 7 gives the pseudo code of the straightforward algorithm for computing the chains of a permutation. The inverse permutation (line 1) can be computed by a single scan of π in $\mathcal{O}(n)$. The chains of the permutation are initialised on line 2 with c_0 containing element 1. Storing the chains as lists or arrays is appropriate. The actual computation of the chains of π (lines 3 to 7) consists of a single iteration through the elements of π. For each element two constant time operations are performed, i.e. a single comparison and appending the element to the correct chain. Therefore, the algorithm has the overall run time $\mathcal{O}(n)$.

Algorithm 7: ComputeChains

Input: permutation π of size n
Output: ordered list of chains $c(\pi)$

```
1 compute π^{-1};
2 c ← ((1));
3 for e = 2 to n do
4     if π^{-1}(e − 1) > π^{-1}(e) then
5         append a new chain (e) to c;
6     else
7         append e to the last chain in c;
8 return c
```

5.3.2 Basic properties of chains

Recall, sorting is done to the identity and note ι is the only permutation which has only one chain. Hence, the problem of sorting by TDRLs corresponds to the subsequent application of a minimum number of TDRLs transforming a given permutation into permutation with only one chain. A TDRL moves the elements of the first copy to the left and the elements of the second copy to the right, such that the order of elements kept in the same copy is not changed. Formally, this is stated in the following proposition.

Proposition 5.3.1. *Let* $\tau = (F, S)$ *be a TDRL, let* π *be a permutation, and let* e_1 *and* e_2 *be two elements of* π*.*

- *If* $e_1 \in F$ *and* $e_2 \in S$ *then* $(\pi \circ \tau)^{-1}(e_1) < (\pi \circ \tau)^{-1}(e_2)$*.*
- *If* $C_\tau(e_1) = C_\tau(e_2)$ *and* $\pi^{-1}(e_1) < \pi^{-1}(e_2)$ *then* $(\pi \circ \tau)^{-1}(e_1) < (\pi \circ \tau)^{-1}(e_2)$*.*

Proof. By the definition of TDRLs the elements kept in the first copy are left of the elements kept in the second copy if a TDRL τ is applied to a permutation π. Furthermore, the element

order of the elements within F and within S is not changed due to τ. Thus, the relative order of elements kept in the same copy is unchanged. □

Changes in the order of the elements of a permutation due to a TDRL have implications on the chains. Some chains might be split whereas other chains might get connected. This is formally described in the following two propositions in more detail.

Proposition 5.3.2. *Let π be a permutation, e_1 and e_2 be two successive elements (i.e. $e_2 = e_1 + 1$) in the same chain c of π, and $\tau = (F, S)$ be a TDRL applied to π. Then chain c is split into a chain ending with e_1 and another chain starting with e_2 in $\pi \circ \tau$ iff $e_1 \in S$ and $e_2 \in F$.*

Proof. As e_1 and e_2 are successive elements in the same chain, $\pi^{-1}(e_1) < \pi^{-1}(e_2)$ holds, i.e. element e_1 is to the left of element e_2. By Proposition 5.3.1, this still holds for $\pi \circ \tau$ if $C_\tau(e_1) = C_\tau(e_2)$ (i.e. e_1 and e_2 are in the same copy of the TDRL) as well as if $e_1 \in F, e_2 \in S$. The remaining case is $e_1 \in S$ and $e_2 \in F$. In this case the elements are not in the same chain in $\pi \circ \tau$, as by Proposition 5.3.1 $(\pi \circ \tau)^{-1}(e_1) > (\pi \circ \tau)^{-1}(e_2)$. Thus, e_1 must be the end of a chain and e_2 will be the start of another chain. More precisely, e_1 and e_2 are in successive chains, i.e. $C_{\pi\circ\tau}(e_1) + 1 = C_{\pi\circ\tau}(e_2)$ holds. □

Proposition 5.3.3. *Let π be a permutation and e_1 and e_2 be two successive elements (i.e. $e_2 = e_1 + 1$) of π which are in different chains. Let $\tau = (F, S)$ be a TDRL applied to π. Elements e_1 and e_2 are in the same chain in $\pi \circ \tau$ iff $e_1 \in F$ and $e_2 \in S$.*

Proof. As elements e_1 and e_2 are successive elements in different chains of π, it holds that $\pi^{-1}(e_1) > \pi^{-1}(e_2)$. By Proposition 5.3.1 this still holds in the case of $C_\tau(e_1) = C_\tau(e_2)$ as well as in the case of $e_1 \in S, e_2 \in F$. In these cases e_1 and e_2 are still in different chains in $\pi \circ \tau$. The remaining case is $e_1 \in F, e_2 \in S$. By Proposition 5.3.1 $(\pi \circ \tau)^{-1}(e_1) < (\pi \circ \tau)^{-1}(e_2)$ holds. Therefore, e_1 and e_2 will be connected in the same chain. □

Summarising Propositions 5.3.2 and 5.3.3, a single TDRL operation can connect successive chains and split others at the same time depending on the set of elements which are kept in the first respectively the second copy. With the propositions that have been presented in this section the effects of a TDRL on the chains of a permutation are completely described.

5.3.3 TDRL distance

In Chaudhuri et al. (2006) it was shown that the TDRL distance $d(\iota, \pi) = \lceil \log_2(\varrho(\pi)) \rceil$, with $\varrho(\pi)$ being the number of maximal increasing substrings in π, i.e. a substring of maximal length with increasing consecutive elements (in Chaudhuri et al. (2006) a permutation π is seen as a string with elements of the permutation as characters). The following proposition clarifies the relation of maximal increasing substrings and chains.

Proposition 5.3.4. *Let π be a permutation of length n. Then $s = \big(\pi(i)\ \pi(i+1)\ \dots\ \pi(k)\big)$ is a maximal increasing substring of π iff $c = (i, i+1, \dots, k)$ is a chain in π^{-1}.*

Proof. The following three preconditions are shown: i) Consecutive positions of elements in π correspond to successive elements in π^{-1}, ii) the property that elements in π are increasing corresponds to the left-to-right ordering of the corresponding elements in π^{-1}, and iii) an increasing substring s of π is maximal iff the corresponding chain c in π^{-1} is maximal. The equivalence given in the proposition follows immediately. Note, i) and ii) hold for π and π^{-1} and are therefore in particular true for string s and chain c. Recall, $\pi^{-1}\big(\pi(j)\big) = j$, therefore it holds that $c = (\pi^{-1}\big(\pi(i)\big), \dots, \pi^{-1}\big(\pi(k)\big))$. The following equivalences hold:

i) Elements $\pi(j)$ and $\pi(j')$ are at consecutive positions in $\pi \Leftrightarrow \pi^{-1}\big(\pi(j)\big) = \pi^{-1}\big(\pi(j')\big) \pm 1$ $\Leftrightarrow j = j' \pm 1 \Leftrightarrow j$ and j' are successive elements in π^{-1}.

ii) Elements $\pi(j)$ and $\pi(j')$ are increasing $\Leftrightarrow \pi(j) < \pi(j') \Leftrightarrow (\pi^{-1})^{-1}(j) < (\pi^{-1})^{-1}(j') \Leftrightarrow$ the position of j in π^{-1} is left from the position of j' in π^{-1}.

iii) Moreover, an increasing string $s = \big(\pi(i)\ \pi(i+1)\ \dots\ \pi(k)\big)$, with $1 \leq i \leq k \leq n$, is maximal, iff the element to the left of $\pi(i)$ in π (if existent) is larger than $\pi(i)$ (this property is called *left-maximality*) and the element to the right of $\pi(k)$ in π (if existent) is smaller than $\pi(k)$ (this is called *right-maximality*). The equivalency for the left-maximality and right-maximality is shown separately in the following.

   a) The increasing string s is left-maximal $\Leftrightarrow \pi(i-1) > \pi(i)$ or $\pi(i)$ is the first element in π (i.e. $i = 1$) $\Leftrightarrow (\pi^{-1})^{-1}(i-1) > (\pi^{-1})^{-1}(i)$ or 1 is the smallest element in $\pi^{-1} \Leftrightarrow$ the element i is located left of element $i-1$ in π^{-1} or $i = 1 \Leftrightarrow$ chain c is left-maximal.

   b) The increasing string s is right-maximal $\Leftrightarrow \pi(i) > \pi(i+1)$ or $\pi(i)$ is the last element in π (i.e. $i = n$) $\Leftrightarrow (\pi^{-1})^{-1}(i) > (\pi^{-1})^{-1}(i+1)$ or n is the largest element in π^{-1} $\Leftrightarrow$ the element i is located right of element $i+1$ in π^{-1} or $i = n \Leftrightarrow$ chain c is right-maximal.

This completes the proof. □

By simply renaming all elements in π it is clear that $d(\iota, \pi) = d(\pi^{-1}, \iota)$. Using the one to one correspondence between maximal increasing substrings in π and chains in π^{-1} as shown in Proposition 5.3.4, it follows that the TDRL distance can be computed by $d(\pi, \iota) = \lceil \log_2(\rho(\pi)) \rceil$, where $\rho(\pi)$ is the number of chains in π. Note, the TDRL distance is not symmetric.

The following proposition states by which amount a TDRL must reduce the number of chains in order to be a sorting TDRL.

Proposition 5.3.5. *Let π be a permutation and τ be a TDRL. Then τ is a sorting TDRL for π iff $\lceil \frac{\rho(\pi)}{2} \rceil \leq \rho(\pi \circ \tau) \leq 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$.*

Proof. The two inequalities are proven separately.

i) It holds that τ is a sorting TDRL for $\pi \Leftrightarrow d(\pi \circ \tau) < d(\pi) \Leftrightarrow \lceil \log_2(\rho(\pi \circ \tau)) \rceil < \lceil \log_2(\rho(\pi)) \rceil \Leftrightarrow \log_2(\rho(\pi \circ \tau)) \leq \lceil \log_2(\rho(\pi)) \rceil - 1 \Leftrightarrow \rho(\pi \circ \tau) \leq 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$.

ii) Assume that $\rho(\pi \circ \tau) < \lceil \frac{\rho(\pi)}{2} \rceil$. Then $d(\pi \circ \tau) < d(\pi) - 1$. This contradicts the TDRL distance.

□

The propositions shows that, in order to be sorting for a permutation π, a TDRL τ has to reduce the number of chains $\rho(\pi)$ to the next smaller value which is a power of 2. That is the number of chains has to be reduced at least by $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$. Furthermore, the proposition shows that the number of chains can not be reduced by more than half.

This completes the set of propositions necessary to study the set of all sorting TDRLs. With Propositions 5.3.2 and 5.3.3 changes in the chains and therefore also changes in the number of chains due to a TDRL can be determined easily. And Proposition 5.3.5 gives the necessary reduction of the number of chains due to a sorting TDRL.

5.4 Restricted TDRLs

Before the set of sorting TDRLs is studied, a subset of TDRLs is introduced and studied in the following. That is only TDRLs are considered for which any existing chain in a permutation is completely included either in the first or in the second copy of a TDRL. Such TDRLs will be referred to as *restricted TDRLs*. The study of the restricted case is useful for the following reasons. First of all, the study of the set of all sorting restricted TDRLs turns out to be easier and will therefore be used as a stepping stone for the general case in Sections 5.5 and 5.6. Secondly — as shown in the following — the distance is not affected by the imposed restriction. Last, but not least, it will be shown that the set of sorting restricted TDRLs is smaller than the set of all sorting TDRLs. Note also that the sorting TDRL scenarios computed with the algorithm of Chaudhuri et al. (2006) consist of restricted TDRLs only.

For a permutation π, the problem of *sorting by restricted TDRLs* is to find a sequence of restricted TDRLs of minimum length that transforms π into the identity. The length of such a sequence is referred to as *restricted TDRL distance*. The set of all sorting restricted TDRLs for a permutation π is denoted by $\mathcal{S}_r(\pi)$. For the restricted case the function $C_\tau : \{c_1, \ldots, c_{\rho(\pi)}\} \mapsto \{1, 2\}$ is defined such that for a TDRL $\tau = (F, S)$ and a chain c of a permutation $C_\tau(c) = 1$ iff $c \in F$ and $C_\tau(c) = 2$ iff $c \in S$.

In order to show that the TDRL distance and the restricted TDRL distance are equal some propositions are necessary. The following propositions describes the effects of restricted TDRLs on the chains of a permutation. One effect of the restriction imposed on the TDRL

model is that existing chains can not be split. This is formally stated in the following proposition.

Proposition 5.4.1. *Let π be a permutation and e_1 and e_2 be two elements of π. If $C_\pi(e_1) = C_\pi(e_2)$ then $C_{\pi\circ\tau}(e_1) = C_{\pi\circ\tau}(e_2)$ holds for every restricted TDRL τ.*

Proof. Let e_1 and e_2 be two elements in the same chain. Then by the definition of restricted TDRLs both elements must be kept in the same copy of the restricted TDRL τ. Thus, by Proposition 5.3.1 the relative order of the elements e_1 and e_2 is not changed and therefore they are in the same chain in $\pi \circ \tau$. □

As by Proposition 5.4.1 chains can not be split with restricted TDRLs it follows that only whole chains can be connected. Formally, a TDRL τ is said to *merge* two chains c and c' of a permutation π if for all $e \in c$ and all $f \in c'$ it holds that $C_{\pi\circ\tau}(e) = C_{\pi\circ\tau}(f)$.

As shown in the previous proposition, restricted TDRLs can not merge and split chains at the same time — in contrast to unrestricted TDRLs. More precisely restricted TDRLs can not split chains at all. The implications on which chains can be merged with a single TDRL are described in the following proposition.

Proposition 5.4.2. *Three chains c_i, c_{i+1}, and c_{i+2} can not be merged with one restricted TDRL.*

Proof. Assume that there is a restricted TDRL τ that merges c_i, c_{i+1}, and c_{i+2}. Then the largest element of c_i has to be kept in the first copy and the smallest element of c_{i+1} has to be kept in the second copy. Furthermore, the largest element of c_{i+1} has to be kept in the first copy and the smallest element of c_{i+2} has to be kept in the second copy. As τ is a restricted TDRL, i.e. each chain has to be included either in the first or the second copy completely. This is a contradiction because the smallest element of c_{i+1} has to be kept in the second copy and the largest element of c_{i+1} has to be kept in the first copy. □

The following specialisation of Proposition 5.3.3 to restricted TDRLs describes how chains can be merged with restricted TDRLs.

Proposition 5.4.3. *Let τ be a TDRL applied to π and let c_i and c_j be chains of π. τ merges c_i and c_j iff $i + 1 = j$ and $C_\tau(c_i) = 1$ and $C_\tau(c_j) = 2$.*

Proof. $\Leftarrow$) Let τ be a TDRL with chain $C_\tau(c_i) = 1$ and chain $C_\tau(c_j) = 2$ with $j = i + 1$. By Proposition 5.4.1 neither chain c_i nor chain c_j is split due to the application of τ. Furthermore, the elements of c_i are to the left of the elements of c_{i+1} in $\pi\circ\tau$ by Proposition 5.3.1, this holds in particular for the biggest element b in c_i and the smallest element s of c_j. As $s = b + 1$ the chains c_i and c_j are merged due to the applied TDRL by Proposition 5.3.3.

$\Rightarrow$) Let τ be a TDRL that merges chains c_i and c_j. First, it is shown that in order to merge chains c_i and c_j by τ, the chain indices must be successive, i.e. $i+1=j$. Second, it is shown that in order to be a TDRL that merges chains c_i and c_j, the elements of chain c_i have to be kept in the first copy of τ and the elements of chain c_j have to be kept in the second copy of τ, i.e. $C_\tau(c_i)=1$ and $C_\tau(c_j)=2$.

i) Assume that $i+1 \neq j$. Then there is no element $e \in c_i$ such that $(e+1) \in c_j$. This contradicts the possibility that the chains are merged as a chain consist of successive elements only. Without loss of generality assume that $i<j$. Then by Proposition 5.4.2 $c_i, c_{i+1}, \ldots, c_{j-1}, c_j$ can not be merged. That is c_i and c_j can not be merged indirectly by merging several chains.

ii) Assume that $C_\tau(c_i)=1$ and $C_\tau(c_j)=2$ does not hold, i.e. the elements of chains c_i and c_j are either both kept in the same copy, or the elements of c_i are kept in the second copy and the elements of c_j are kept in the first copy, i.e. $C_\tau(c_i)=2$ and $C_\tau(c_j)=1$. If the two elements of the chains are kept in the same copy, the relative order of the union of elements of both chains is not changed due to Proposition 5.3.1. If the elements of c_i are kept in the second copy and the elements of c_j are kept in the first copy, all the elements of c_i are to the right of all elements of c_j after applying the TDRL τ. In both cases the chains are not merged due to τ, which is a contradiction.

Thus, the only way to merge the chains with restricted TDRLs is the one given in the proposition. □

Using Propositions 5.4.2 and 5.4.3 the following theorem shows that the restricted TDRL distance is identical to the general TDRL distance.

Theorem 5.4.4. *For a permutation π the restricted TDRL distance is given by*

$$d(\pi) = \lceil \log_2(\rho(\pi)) \rceil.$$

Proof. Firstly, $d(\pi,\iota) \leq \lceil \log_2(\rho(\pi)) \rceil$ is shown, secondly, $d(\pi,\iota) \geq \lceil \log_2(\rho(\pi)) \rceil$ is shown.

i) Let $c_1, \ldots, c_{\rho(\pi)}$ be the chains of permutation π. Let τ be a restricted TDRL with $c_{2i-1} \in F$ and $c_{2i} \in S$, $i \in [1 : \lfloor \frac{\rho(\pi)}{2} \rfloor]$. Due to the pairwise merging of the chains c_{2i-1} and c_{2i} it is easy to see that $\rho(\pi \circ \tau) = \lceil \frac{\rho(\pi)}{2} \rceil$ holds. Therefore, the restricted TDRL distance is at most $\lceil \log_2(\rho(\pi)) \rceil$.

ii) By Propositions 5.4.3 and 5.4.2 it is not possible to merge more than two successive chains with one restricted TDRL. Hence, $d(\pi,\iota) \geq \lceil \log_2(\rho(\pi)) \rceil$ holds.

□

5.5 All sorting TDRLs

In the following sections it is shown how all sorting TDRLs are computed. The methods described in this section are based on Bernt et al. (2009b). First, the case of restricted TDRLs is considered in Section 5.5.1. The techniques developed for the restricted case are applied in Section 5.5.2 to the more complicated general case. A closed formula for the number of all sorting restricted TDRLs is given and an algorithm for enumerating all sorting TDRLs is inferred.

5.5.1 The restricted case

The problem of computing the number of sorting restricted TDRL for a permutation π (toward ι) can be rephrased as a combinatorial problem of binary strings over the alphabet $\Sigma = \{1, 2\}$. Let $\tau = (F, S)$ be a restricted TDRL for a permutation π and $t = t(1) \dots t(\rho(\pi))$ be a binary string of length $\rho(\pi)$ with $t(i) = C_\tau(c_i)$. Thus, the string t corresponds to a restricted TDRL, such that the i-th element of t indicates if the i-th chain is kept in the first or in the second copy of τ. Recall, by Proposition 5.4.3 only successive chains c_{i-1} and c_i, with $i \in [2 : \rho(\pi)]$, can be merged with restricted TDRLs. Furthermore, this is achieved only by keeping c_{i-1} in the first copy and c_i in the second copy. This property is translated to the binary string representation of restricted TDRLs in the following way. Clearly, only consecutive positions in the binary string have to be regarded. Moreover two chains c_{i-1} and c_i ($i \in [2 : \rho(\pi)]$) are merged by a TDRL iff the corresponding binary string has a 12-transition at position i. Thus, the number of pairs of chains that are merged by a restricted TDRL, i.e. the reduction of the number of chain, is equal to the number of 12-transitions in the corresponding binary string. Thus, by Proposition 5.3.5 a binary string of length $\rho(\pi)$ corresponds to a sorting restricted TDRL iff the number of 12-transitions is at least $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$. Therefore the number of sorting restricted TDRLs for a permutation π corresponds to the number of binary strings for which the number of 12-transitions is at least k with $k = \rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$. The following lemma gives the number of possible binary strings of length n with k 12-transitions.

Lemma 5.5.1. *The number of possible binary strings of length n over the alphabet $\Sigma = \{1, 2\}$, which have k 12-transitions, is*

$$\binom{n+1}{2k+1}$$

Proof. The four possible cases for choosing the first and last character in t are considered individually. Between two consecutive 12-transitions, a 21-transition has to occur. Hence, in the case $t(1) = t(n) = 1$ or $t(1) = t(n) = 2$ there are $\binom{n-1}{2k}$ possibilities to place k 12-transitions. This is because the position for $2k$ alternating transitions have to be chosen out

of the $n-1$ possible positions in order to get k 12-transitions. For strings starting with 1 and ending with 2, only $2k-1$ positions for the transitions have to be chosen. Thus, there are $\binom{n-1}{2k-1}$ such strings. In the remaining case of $t(1)=2$ and $t(n)=1$ the number of possible strings is $\binom{n-1}{2k+1}$ as the position of one additional transition has to be chosen in order to have $t(n)=1$. The sum of the four binomials is:

$$\binom{n-1}{2k-1}+\binom{n-1}{2k}+\binom{n-1}{2k}+\binom{n-1}{2k+1}=\binom{n}{2k}+\binom{n}{2k+1}=\binom{n+1}{2k+1}.$$

□

The number of possible binary string with at least k 12-transitions follows immediately as the sum over all values between k and the maximal number of 12-transitions.

Corollary 5.5.2. *The number of possible binary strings of length n over the alphabet $\Sigma=\{1,2\}$, which have at least k 12-transitions, is*

$$\sum_{i=k}^{\lfloor\frac{n}{2}\rfloor}\binom{n+1}{2i+1}$$

The upper bound of the summation is because there exists no string of length n with more than $\frac{n}{2}$ 12-transitions but choosing a larger upper bound does not invalidate the corollary.

Based on Corollary 5.5.2 the number of sorting restricted TDRLs follows.

Corollary 5.5.3. *For a permutation π with $\rho(\pi)$ chains the number of sorting restricted TDRLs is*

$$|\mathcal{S}_r(\pi)|=\sum_{i=\rho(\pi)-2^{\lceil\log_2(\rho(\pi))\rceil-1}}^{\lfloor\frac{\rho(\pi)}{2}\rfloor}\binom{\rho(\pi)+1}{2i+1}.$$

Proof. Let π be a permutation with $\rho(\pi)$ chains, i.e. the TDRL distance is $\lceil\log_2(\rho(\pi))\rceil$. A restricted TDRL applied to π is sorting iff it reduces the number of chains from $\rho(\pi)$ at least to the largest number of chains corresponding to $d(\pi)-1$, i.e. $2^{d(\pi)-1}=2^{\lceil\log_2(\rho(\pi))\rceil-1}$ (see proposition 5.3.5). A binary string corresponding to a restricted TDRL for π has length $\rho(\pi)$. If a binary string has at least $k=\rho(\pi)-2^{\lceil\log_2(\rho(\pi))\rceil-1}$ 12-transitions, it corresponds to a sorting restricted TDRLs. By substituting n and k in Corollary 5.5.2 the result follows immediately. □

Interestingly, there exists only one sorting restricted TDRL if the number of chains $\rho(\pi)$ is equal to a power of 2. Also there is only one possible sequence of sorting TDRLs towards ι because after applying the TDRL the number of chains is halved and is therefore still a power of 2. If the number of chains is $\rho(\pi)=2^k-1$ for some k, there exist several sorting restricted

TDRLs, but the subsequent TDRLs towards ι are predetermined as $\lceil (2^k - 1)/2 \rceil = 2^{k-1}$. This is stated in the following proposition.

Proposition 5.5.4. *Let π be a permutation with $\rho(\pi)$ chains. After the application of $\lceil \log_2(2^{\lceil \log_2(\rho(\pi)) \rceil} - \rho(\pi) + 1) \rceil$ sorting restricted TDRLs, the number of remaining chains is a power of 2, and therefore the remaining sorting sequence towards ι has no alternatives.*

Proof. Assume that a variable is initialised with an integer $\rho > 1$ and let an operation reduce the value of the variable from ρ to ρ' with $\lceil \rho/2 \rceil \leq \rho' \leq 2^{\lceil \log_2(\rho) \rceil - 1}$. Observe, this is a sorting TDRL's effect on the number of existing chains. The worst case, with respect to the difference of ρ' to the next larger power of 2, is if $\rho' = \lceil \rho/2 \rceil$. In this case the difference of ρ' to the next larger power of 2 is $2^{\lceil \log_2(\rho') \rceil} - \rho' = 2^{\lceil \log_2(\lceil \rho/2 \rceil) \rceil} - \lceil \rho/2 \rceil = 2^{\lceil \log_2(\rho) \rceil - 1} - \lceil \rho/2 \rceil = \lceil \frac{2^{\lceil \log_2(\rho) \rceil} - \rho}{2} \rceil$. That is in the worst case the difference to the next bigger power of 2 is halved. Therefore, in the worst case, after $\lceil \log_2(2^{\lceil \log_2(\rho(\pi)) \rceil} - \rho(\pi) + 1) \rceil$ divisions by 2 a power of 2 is reached. □

Note that many TDRLs reduce the number of chains to a power of 2 in less steps. For example there is always at least one TDRL that reduces the number of chains to the next smaller power of 2.

5.5.2 The general case

Also for the general case of computing all sorting TDRLs the problem can be formulated as a problem of finding binary strings with certain properties. First some definitions are necessary.

Similar to the case of restricted TDRLs a (general) TDRL can be translated to a binary string. But in contrast to the restricted case, where each position in the binary string defines in which copy a complete chain is kept, the binary string corresponding to a general TDRL must define for each element in which copy it is kept. Let τ be a TDRL applied to a permutation of size n. The binary string $t = t(1) \dots t(n)$ of length n over the alphabet $\{1, 2\}$ with $t(i) = C_\tau(i)$, $i \in [1 : n]$, corresponds to τ.

The information given by string t, i.e. in which copy an element is kept, is not sufficient to describe the effect of a TDRL on a permutation π, but the additional information if the corresponding elements are in the same chain of π or not is necessary. In the restricted case this additional information is given implicitly. Let π be a permutation of length n. Let $p = p(1) \dots p(n-1)$ be a binary string of length $n-1$ over the alphabet $\{\phi, \theta\}$ defined by the chains of π as follows. If $C_\pi(e) = C_\pi(e+1)$ (i.e. if element e and $e+1$ are in the same chain), $p(e) = \theta$, otherwise $p(e) = \phi$, $e \in [1 : n-1]$. String p is called *transition string* of π. Note, p depends only on (the chains of) the permutation π and t depends only on τ.

By Propositions 5.3.2 and 5.3.3 the effects of a TDRL τ on the chains of a permutation π are defined by the corresponding strings t and p. If $t(e) = 2$, $t(e+1) = 1$, and $p(e) = \theta$ (denoted as a 21_θ-transition), then the chain with index $C_\pi(e)$ is split after element e and the

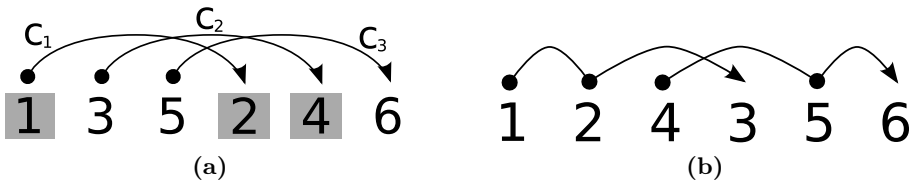


Figure 5.2 – a) Permutation π with three chains c_1, c_2, c_3; the corresponding string $p = \theta\phi\theta\phi\theta$, e.g. $p(3) = \theta$ as $C_\pi(3) = C_\pi(4) = 2$; one of the sorting TDRLs is $\tau = (\{1, 2, 4\}, \{3, 5, 6\})$ where the elements from F are boxed and elements from S not; the corresponding $t = 112122$; b) Permutation $\pi \circ \tau$: elements 2 and 3 are connected due to τ, as $t(2) = 1$, $t(3) = 2$, and $p(2) = \phi$ induce a 12_ϕ-transition; furthermore, τ connects elements 4 with 5 and destroys chain $c_2 = (3, 4)$

number of chains is increased by one. If $t(e) = 1$, $t(e + 1) = 2$, and $p(e) = \phi$ (12_ϕ-transition) then the number of chains is decreased by one as the element e gets connected to element $e + 1$. See Figure 5.2 for an illustration.

Let Φ be the number of 12_ϕ-transitions in t, and let Θ be the number of 21_θ-transitions in t. Applying a TDRL τ (corresponding to a binary string t) to a permutation π with transition string p, reduces the number of chains by Φ and increases it by Θ. Therefore, the overall reduction in the number of chains due to τ is $k = \Phi - \Theta$. By Proposition 5.3.5, the computation of the number of sorting TDRLs for a given permutation π is equivalent to the computation of the number of strings t (corresponding to a TDRL τ) such that k is between $\lfloor \frac{\rho(\pi)}{2} \rfloor$ and $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$.

Let a transition string p be given. Let a *k-sorting string* be a binary string t, for which the number of 12_ϕ-transitions is Φ, the number of 21_θ-transitions is Θ, and $k = \Phi - \Theta$. The number of TDRLs for a permutation π, reducing the number of chains by k, is equivalent to the number of k-sorting strings t with p being the transition string of π.

For the computation of the number of sorting TDRLs a dynamic programming scheme is applied as follows. Let $a^x_{j,k}$ be the number of k-sorting strings t of length j ending with $x \in \{1, 2\}$. All values of the dynamic programming matrix are initialised with 0 except for $a^1_{1,0} = 1$ and $a^2_{1,0} = 1$. With the following recursion $a^x_{n,k}$ can be computed. An illustration of the recursion for the dynamic programming is given in Figure 5.3a.

$$a^1_{j+1,k} = \begin{cases} a^1_{j,k} + a^2_{j,k+1} & \text{if } p(j) = \theta \\ a^1_{j,k} + a^2_{j,k} & \text{else} \end{cases} \tag{5.1}$$

$$a^2_{j+1,k} = \begin{cases} a^1_{j,k-1} + a^2_{j,k} & \text{if } p(j) = \phi \\ a^1_{j,k} + a^2_{j,k} & \text{else} \end{cases} \tag{5.2}$$

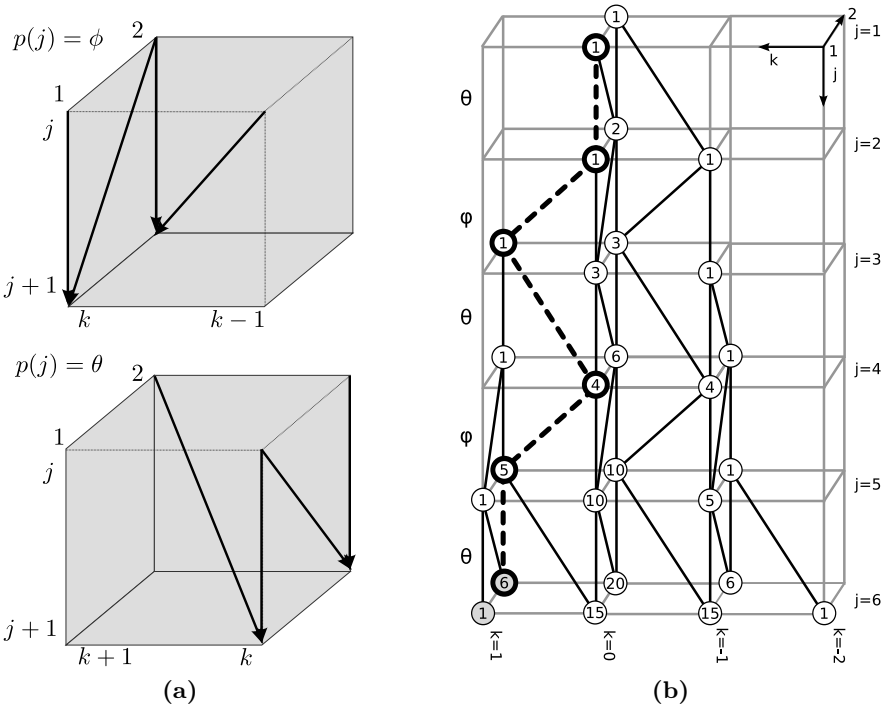


Figure 5.3 – a) Illustration of the recursion in the dynamic programming matrix for $p(j) = \phi$ (top) and $p(j-1) = \theta$ (bottom); arrows indicate which values are used to compute the sums of the recursion; b) the dynamic programming matrix for the example in Figure 5.2; values in circles correspond to $a^x_{j,k}$; the sum of all values in grey filled circles corresponds to the number of all sorting TDRLs; the dashed path corresponds to the TDRL $\tau = (\{1,2,4\},\{3,5,6\})$

In order to compute the number of k-sorting strings of length $j+1$ ending with x the type of the transition at position $j+1$ and the value of the string p at position j has to be regarded. More precisely, the cases where the addition of a new element leads to a new 12_ϕ- or 21_θ-transition must gain special attention. These cases are:

i) if $p(j) = \theta$ then appending a 1 to any k-sorting string of length j ending with 2 generates a $(k-1)$-sorting string because a new 21_θ-transition is added,

ii) if $p(j) = \phi$ then appending a 2 to any k-sorting string of length j ending with j generates a $(k+1)$-sorting string because a new 12_ϕ-transition is added.

In the remaining cases appending a new element to any k-sorting string generates also a k-sorting string. The base cases of the recursion, i.e. $a^1_{1,0} = 1$ and $a^2_{1,0} = 1$, stem from the fact that there is only one binary string of length 1 ending with 1 (respectively 2) and that this string is a 0-sorting string.

Note, although valid from a combinatorial point of view, $a^x_{j,k}$ with $j < n$ does not correspond to a number of sorting TDRLs, as strings t of size $j < n$ do not define valid TDRLs.

The number of sorting TDRLS follows immediately.

Corollary 5.5.5. *For a permutation π of length n with $\rho(\pi)$ chains the number of sorting TDRLs is*

$$|\mathcal{S}(\pi)| = \sum_{i=\rho(\pi)-2^{\lceil \log_2(\rho(\pi))\rceil -1}}^{\lfloor \frac{\rho(\pi)}{2} \rfloor} \left(a^1_{n,i} + a^2_{n,i}\right).$$

The set of all sorting TDRLs can be inferred directly from the dynamic programming matrix by backtracking. That is every path, consisting of $n-1$ edges from an $a^x_{n,k}$, with $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi))\rceil -1} \leq k \leq \lfloor \frac{\rho(\pi)}{2} \rfloor$, to $a^x_{1,0}$, $x \in \{1,2\}$, corresponds to a sorting TDRL (for an example see Figure 5.3).

5.6 A new method for computing all sorting TDRLs

The methods described in the last section have been implemented in order to generate the results for Bernt et al. (2009b). During the generation of the results it was observed that the number of sorting TDRLs apparently depends only on the number of chains of a permutation and not on the position of the elements of the chains in the permutation. This was surprising and suggested the exciting possibility to derive a closed formula for the general case, too. The values that are computed during the dynamic programming have been investigated in more detail. The following observations have been made:

i) apparently the j-th row of the dynamic programming matrix always contains the values of the j-th row of Pascal's triangle regardless of the values of $p(i)$, $i \in [1 : j-1]$ (see for an example Figure 5.3b), and
ii) the values $p(i)$, $i \in [1 : j-1]$ influence the column, i.e. the k where the first non-zero value can be found in the n-th row.

From these observations it was conjectured that the number of sorting TDRLs can be computed as a prefix sum of the n-th row of Pascal's triangle. A similar conjecture was derived for the number of restricted TDRLs. The correctness of these conjectures was shown by proving the equivalence to the results presented in the last section. This was already a fascinating result but not completely satisfying because the combinatorial interpretation of these, mathematically appealing, results have been missing and one may "suspect that a much simpler algorithm will do the trick"[1].

In this section the missing combinatorial interpretation and alternative (and simpler) methods are presented for the first time. Like in the last section first the result for the case of restricted TDRLs are presented in Section 5.6.1 and then, building on this results, the general case is presented in Section 5.6.2. The equivalence of the results is shown in Section 5.7.

[1] Comment of an anonymous reviewer

			# of trans.	
E	T	s	12	21
$\emptyset$	$\{1,\dots,5\}$	$2 \bullet 1 \bullet 2 \bullet 1 \bullet 2 \bullet 1$	2	2(3)
$\{1\}$	$\{2,3,4,5\}$	$2 \circ 2 \bullet 1 \bullet 2 \bullet 1 \bullet 2$	2	2(2)
$\{4\}$	$\{1,2,3,5\}$	$2 \bullet 1 \bullet 2 \bullet 1 \circ 1 \bullet 2$	2	1(2)
		$\vdots$		
$\{1,3\}$	$\{2,4,5\}$	$2 \circ 2 \bullet 1 \circ 1 \bullet 2 \bullet 1$	1	2(2)
$\{2,5\}$	$\{1,3,4\}$	$2 \bullet 1 \circ 1 \bullet 2 \bullet 1 \circ 1$	1	1(2)
		$\vdots$		
$\{1,3,5\}$	$\{2,4\}$	$2 \circ 2 \bullet 1 \circ 1 \bullet 2 \circ 2$	1	1(1)
$\{2,4,5\}$	$\{1,3\}$	$2 \bullet 1 \circ 1 \bullet 2 \circ 2 \circ 2$	1	0(1)

Table 5.1 – Examples for strings of length 5 that are recursively defined by different bipartitions (T, E) of $\{1,\dots,n\}$ and the corresponding numbers of 12- and 21-transitions; columns E and T give the bipartition used to recursively define the string s given in the third column; note that in the strings the additional element $s(0) = 2$, that does not belong to s, is also given in grey; a $\circ$ between elements $s(i-1)$ and $s(i)$ indicates that $i \in E$ and a $\bullet$ indicates that $i \in T$; the last two columns give the number of 12- and 21-transitions in s (the number of 21-transitions in $s(0)s(1)\dots s(5)$ is given in parentheses)

5.6.1 The restricted case

Similar to Section 5.5.1 the new method presented in this section is based on the insight that all sorting restricted TDRLs can be found by enumerating binary strings with a certain number of 12-transitions. Hence, in the first part of this section the number of binary strings of length n with at least k 12-transitions is derived. In contrast to Section 5.5.1 the recursive definition of a binary string, i.e. by the specification of the positions where a transition takes place, respectively no transition takes place, is used. Let (T, E) be a bipartition of $\{1,\dots,n\}$ where T specifies the positions where a transition takes place and E specifies the positions where no transition takes place. Some examples for binary strings of length n that are recursively defined by different bipartitions (T, E) of $\{1,\dots,n\}$ are given in Table 5.1. Apparently the number of 12-transitions is related to the size of the set T (respectively E). This relation is given formally in the following lemma.

Lemma 5.6.1. *Let (T, E) be a bipartition of $\{1,\dots,n\}$ and $t = t(1)\dots t(n)$ be the binary string over $\Sigma = \{1, 2\}$ of length n that is recursively defined by (T, E) such that $t(i) \neq t(i-1)$ iff $i \in T$ (respectively $t(i) = t(i-1)$ iff $i \in E$) with $i \in [1 : n]$ and $t(0) = 2$. Then the number of 12-transitions in t is $\lfloor \frac{|T|}{2} \rfloor = \lfloor \frac{n-|E|}{2} \rfloor$.*

Proof. Let (T, E) be a bipartition of $\{1, \ldots, n\}$ and let $t = t(1)\ \ldots\ t(n)$ denote the string which is recursively defined by T, such that $t(i) \neq t(i-1)$ iff $i \in T$ (respectively $t(i) = t(i-1)$ iff $i \in E$), with $i \in [1:n]$ and $t(0) = 2$. Furthermore, let t' denote the string which includes the element $t(0)$, i.e. $t' = t(0)\ t(1)\ \ldots\ t(n)$. By the recursive definition of t (t') there are $|T|$ transitions in t'. Hence, if $|T|$ is even, the number of 12-transitions is equal to the number of 21-transitions in t', i.e. $\frac{|T|}{2} = \lfloor\frac{|T|}{2}\rfloor$. In the other case where $|T|$ is odd, there is one 12-transition less than 21-transitions. This is because by definition $t(0) = 2$ and therefore the first transition in t' is a 21-transition. Thus, the number of 12-transitions in t' is $\lfloor\frac{|T|}{2}\rfloor$, and the number of 21-transitions in t' is $\lceil\frac{|T|}{2}\rceil$. Again, because by definition $t(0) = 2$, the first transition in t' is a 21-transition, i.e. $(t(0), t(1))$ can not be a 12-transition. Ergo, the number of 12-transitions in t is the same as in t'. That is the number of 12-transitions in t is $\lfloor\frac{|T|}{2}\rfloor$. Obviously $|T| = n - |E|$. This completes the proof. □

Lemma 5.6.1 shows that the number of 12-transitions in a binary string that is recursively defined by a bipartition (T, E) — which specifies the positions where transitions take place or not — does only depend on the size of the sets E and T and not on their contents. Based on Lemma 5.6.1 the following corollary gives a closed formula for the number of binary string with at least k 12-transition.

Corollary 5.6.2. *The number of possible binary strings over $\Sigma = \{1, 2\}$ of length n, which have at least k 12-transitions, is*

$$\sum_{i=0}^{n-2k} \binom{n}{i} = \sum_{i=2k}^{n} \binom{n}{i}.$$

Proof. As a consequence of Lemma 5.6.1 it holds that binary stings of length n which are recursively defined by the sets of a bipartition (T, E) of $\{1, \ldots, n\}$ with $2k \leq |T| \leq n$ have at least k 12-transitions. The number of binary strings of length n that can be defined recursively with a set T of size $|T|$ is equal to the number of possibilities to choose $|T|$ elements out of $\{1, \ldots, n\}$ which is equal to $\binom{n}{|T|}$. Hence, the sum of the number of possibilities to choose the i elements of the set T out of n elements with $i \in [2k:n]$ gives the number of binary strings of length n with at least k 12-transitions.

Likewise, the sum of the number of possibilities to choose the j elements of the set E out of n elements with $j \in [0:n-2k]$ gives the number of binary strings of length n with at least k 12-transitions. Because (T, E) is a bipartition of $\{1, \ldots, n\}$ the equivalence follows from the symmetry of the binomial coefficient: $\binom{n}{|T|} = \binom{n}{n-|T|} = \binom{n}{|E|}$. □

Based on Corollary 5.6.2, in the following corollary the number of sorting restricted TDRLs is given.

Corollary 5.6.3. *For a permutation π with $\rho(\pi)$ chains there are*

$$|\mathcal{S}_r(\pi)| = \sum_{i=0}^{2^{\lceil \log_2(\rho(\pi)) \rceil} - \rho(\pi)} \binom{\rho(\pi)}{i} = \sum_{i=2\rho(\pi)-2^{\lceil \log_2(\rho(\pi)) \rceil}}^{\rho(\pi)} \binom{\rho(\pi)}{i}$$

sorting restricted TDRLs.

Proof. By Proposition 5.3.5 the number of 12-transitions in a binary string t of length $\rho(\pi)$ corresponding to a sorting TDRL τ for a permutation π must be at least $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1}$. Substituting, n with $\rho(\pi)$ and k as above, in Corollary 5.6.2 yields the result. □

Corollary 5.6.3 shows that the number of sorting restricted TDRLs corresponds to a prefix sum in the $\rho(\pi)$-th row of the Pascal's triangle. The presented result also covers perfectly the symmetry of Pascals triangle, i.e. there is a corresponding suffix sum of the $\rho(\pi)$-th row that gives the same value. Note, this sum does only depend on the number of chains and not on the length of the chains, their position in the permutation, or the length of the permutation (apart from the fact that $\rho(\pi) \leq n$).

An algorithm for enumerating all sorting restricted TDRLs for a permutation π can be obtained by enumerating the TDRLs corresponding to the possible binary string of length $\rho(\pi)$ with i 12-transitions for each $i \in [\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1} : \lfloor \frac{\rho(\pi)}{2} \rfloor]$. Note that the enumeration of all binary strings of length $\rho(\pi)$ with i 12-transitions can be realised by enumerating all possibilities to choose a set T of size $2i$ and $2i+1$ out of the set $\{1, \ldots, \rho(\pi)\}$ and printing the binary string which is recursively defined for each T (alternatively this can be done by choosing the set E of size $\rho(\pi) - 2i$ and $\rho(\pi) - (2i+1)$).

5.6.2 The general case

Recall the consequences of a TDRL τ on the chains of a permutation π given in Propositions 5.3.2 and 5.3.3. Two successive elements e and $f = e + 1$, which are in different chains of π, are in the same chain in $\pi \circ \tau$ iff $C_\tau(e) = 1$ and $C_\tau(f) = 2$, thus the number of chains is reduced by such a TDRL. In the case that the two elements e and $f = e + 1$ are in the same chain of π they are in different chains in $\pi \circ \tau$ iff $C_\tau(e) = 2$ and $C_\tau(f) = 1$, hence, the number of chains is increased.

Again, the correspondence between binary strings and TDRLs is used for studying the set of sorting TDRLs. Let τ be a TDRL and t be the corresponding binary string of length n over $\Sigma = \{1, 2\}$ with $t(e) = C_\tau(e)$. That is the string t corresponds to a TDRL, such that $t(e)$ indicates if the corresponding element e of π is kept in the first or in the second copy of τ. For a permutation π let $\theta(\pi) = \{i : C_\pi(i-1) = C_\pi(i), i \in [2 : n]\}$ denote the set of elements whose predecessor is in the same chain of π and $\phi(\pi) = \{i : C_\pi(i-1) \neq C_\pi(i), i \in [2 : n]\} \cup \{1\}$ denote the set of elements whose predecessor is in a different chain of π plus the element

1 (in other words $\phi(\pi)$ is the set of elements that are the first element of a chain). Note, $|\phi(\pi)| = \rho(\pi)$. Furthermore, remark that $(\phi(\pi), \theta(\pi))$ defines a bipartition of $\{1, \ldots, n\}$. The symbols ϕ and θ have been chosen intentionally to indicate the similarity between the transition string as used in Section 5.5.2 and the bipartition $(\phi(\pi), \theta(\pi))$ as defined above. That is $p(i)$ in a transition string is ϕ if $i + 1 \in \phi(\pi)$ and $p(i)$ is equal to θ if $i + 1 \in \theta(\pi)$.

The effects of a TDRL τ, given by Propositions 5.3.2 and 5.3.3, on the chains of a permutation π can be defined by the corresponding string t and the bipartition $(\phi(\pi), \theta(\pi))$. If $t(e-1) = 2$, $t(e) = 1$, and $e \in \theta(\pi)$ (i.e. a 21_θ-*transition*), then and only then the chain with index $C_\pi(e)$ is split after element $e - 1$ and the number of chains is increased by one. If $t(e-1) = 1$, $t(e) = 2$, and $e \in \phi$ (i.e. a 12_ϕ-*transition*) then and only then the number of chains is decreased by one as the element $e - 1$ gets connected to element e.

Let Φ be the number of 12_ϕ-transitions in t, and let Θ be the number of 21_θ-transitions in t. Applying a TDRL τ (corresponding to a binary string t) to a permutation π, reduces the number of chains by Φ and increases it by Θ. Therefore, the overall reduction in the number of chains due to τ is $k = \Phi - \Theta$. The computation of the number of sorting TDRLs for a given permutation π is equivalent to the computation of the number of binary strings t (corresponding to a TDRL τ) such that $\rho(\pi) - 2^{\lceil \log_2(\rho(\pi)) \rceil - 1} \leq k \leq \lfloor \rho(\pi)/2 \rfloor$.

Given a set $M \subseteq \{1, \ldots, n\}$ a binary string of length n that corresponds to a TDRL for a permutation π of size n is recursively defined in the following by

$$t(i) \neq t(i-1) \text{ iff } (i \in \phi(\pi) \wedge i \notin M) \vee (i \notin \phi(\pi) \wedge i \in M). \tag{5.3}$$

The base case is $t(0) = 2$, where again $t(0)$ is not part of the string but only used for the recursive definition.

So far, binary strings have been recursively defined by a set T, i.e. there is a transition at a position i if $i \in T$ and no transition if $i \notin T$. Now a set M and an additional "modifier set" $\phi(\pi)$ is used in Equation (5.3) that reverses the effect of being element in M, i.e.

i) despite $i \in M$ there is no transition at position i if $i \in \phi(\pi)$ and
ii) despite $i \notin M$ there is a transition at position i if $i \in \phi(\pi)$.
iii) Otherwise, if $i \notin \phi$ then M defines the transitions as usual, i.e. there is a transition if $i \in M$ and no transition if $i \notin M$.

Example 5.6.1. *Consider the permutation* $\pi = (1\ 3\ 5\ 2\ 4\ 6)$. π *has the* 3 *chains* $(1, 2)$, $(3, 4)$, *and* $(5, 6)$ *as shown in Figure 5.2. Thus* $\phi(\pi) = \{1, 3, 5\}$ *and* $\theta(\pi) = \{2, 4, 6\}$. *Table 5.2 shows examples of binary strings that are recursively defined by different subsets of* $\{1, \ldots, n\}$ *with the rule of Equation* (5.3).

The first row ($M = \emptyset$) *corresponds to the restricted TDRL* $\tau = (\{1, 2, 5, 6\}, \{3, 4\})$ *that merges chains* c_1 *and* c_2 *and reduces the number of chains by one. Likewise the row for*

M	s	$\Phi - \Theta$
$\emptyset$	2∘1∘1∘2∘2∘1∘1	$1 - 0 = 1$
$\{1\}$	2•2∘2∘1∘1∘2∘2	$1 - 0 = 1$
$\{4\}$	2∘1∘1∘2•1∘2∘2	$2 - 1 = 1$
$\{1,3\}$	2•2∘2•2∘2∘1∘1	$0 - 0 = 0$
$\{2,4\}$	2∘1•2∘1•2∘1∘1	$0 - 0 = 0$
$\{3,6\}$	2∘1∘1•1∘1∘2•1	$1 - 1 = 0$

Table 5.2 – Binary strings that are recursively defined by $\phi(\pi) = \{1, 3, 5\}$ (Example 5.6.1) and different sets M following the rule given in Equation (5.3); M: a subset of $\{1, \ldots, 6\}$; s: the recursively defined string, a $\bullet$ between elements $s(i-1)$ and $s(i)$ indicates that $i \in M$ and a $\circ$ indicates that $i \notin M$, grey columns mark $i \in \phi(\pi)$, note that in the strings the additional element $s(0) = 2$, that does not belong to s, is also given in grey; the last column gives $\Phi - \Theta$

$M = \{3, 6\}$ corresponds to the TDRL $\tau = (\{1, 2, 3, 4, 6\}, \{5\})$ that connects elements 4 and 5 and disconnects 5 and 6 and therefore does not change the number of chains.

Apparently there seems to be a relation of the size of the set M and the value of $\Phi - \Theta$. This relation is stated formally in the following Lemma. But first note that Equation (5.3) is equivalent to $t(i) = t(i-1)$ iff $i \in \phi(\pi) \triangle M$, where $\triangle$ denotes the symmetric difference (for two sets A and B, $A \triangle B = (A \cup B) \setminus (A \cap B) = (A \setminus B) \cup (B \setminus A)$).

Lemma 5.6.4. *Let $M \subseteq \{1, \ldots, n\}$ and (ϕ, θ) be a bipartition of $\{1, \ldots, n\}$ with $1 \in \phi$. Let $s = s(1) \ \ldots \ s(n)$ be the binary string of length n over the alphabet $\Sigma = \{1, 2\}$ that is recursively defined by the set $\phi \triangle M$ such that $s(i) \neq s(i-1)$ iff $i \in \phi \triangle M$ with $i \in [1 : n]$ and $s(0) = 2$. Let Φ be the number of 12_ϕ-transitions in t, and Θ be the number of 21_θ-transitions in t. Then*

$$\Phi - \Theta = \left\lfloor \frac{|\phi| - |M|}{2} \right\rfloor .$$

Proof. A transition in a string t at position i with $t(i-1) = x$ and $t(i) = y$ is denoted as xy_ϕ-transition iff $i \in \phi$ and xy_θ-transition iff $i \in \theta$. Let $T^z_{xy}(t)$, $x, y \in \Sigma$ and $z \in \{\phi, \theta\}$, denote the number of xy_z transitions in the binary string t.

Now let $M \subseteq \{1, \ldots, n\}$ and (ϕ, θ) be a bipartition of $\{1, \ldots, n\}$ with $1 \in \phi$. Let $t = t(1) \ \ldots \ t(n)$ be the binary string of length n over the alphabet $\Sigma = \{1, 2\}$ that is recursively defined by the set $\phi \triangle M$ such that $t(i) \neq t(i-1)$ iff $i \in \phi \triangle M$ with $i \in [1 : n]$ and $t(0) = 2$. Let t' denote the binary string $t(0) \ t(1) \ \ldots \ t(n)$. See Figure 5.4 for an illustration of the relation of the sets M, ϕ, and $\phi \triangle M$ as well as the type of the transition defined by the different sets.

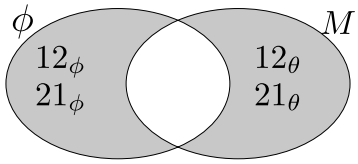


Figure 5.4 – Venn diagram of sets $M \subseteq \{1, \ldots, n\}$ and the set ϕ of a bipartition (ϕ, θ) with $1 \in \phi$. The set $\phi \triangle M$, that defines the transitions of the string t, corresponds to the grey shaded area. The remaining white area (including the outside) correspond to position in the string where no transition takes place. Transitions defined by elements of $\phi \setminus M$ are either 12_ϕ- or 21_ϕ-transitions (the left grey area) and those transitions defined by $M \setminus \phi$ are either 12_θ- or 21_θ-transitions (the right grey area)

The total number of transitions in t' is $|\phi \triangle M|$, hence

$$T_{12}^{\phi}(t') + T_{12}^{\theta}(t') + T_{21}^{\phi}(t') + T_{21}^{\theta}(t') = |\phi \triangle M| \tag{5.4}$$

Because $1 \in \phi$ there is a transition in t' at position 1 iff $1 \notin M$. In this case the transition must be a 21_ϕ-transition because by definition $t(0) = 2$. In any case, i.e. if $1 \in M$ or if $1 \notin M$, it holds $T_{12}^{\phi}(t) = T_{12}^{\phi}(t')$, $T_{21}^{\theta}(t) = T_{21}^{\theta}(t')$, and $T_{12}^{\theta}(t) = T_{12}^{\theta}(t')$. Hence, Equation (5.4) can be transformed into

$$\begin{aligned}
T_{12}^{\phi}(t) + T_{21}^{\theta}(t) &= |\phi \triangle M| - T_{21}^{\phi}(t') - T_{12}^{\theta}(t) \\
T_{12}^{\phi}(t) - T_{21}^{\theta}(t) &= |\phi \triangle M| - T_{21}^{\phi}(t') - T_{12}^{\theta}(t) - 2 \cdot T_{21}^{\theta}(t) \\
T_{12}^{\phi}(t) - T_{21}^{\theta}(t) &= |\phi \triangle M| - (T_{21}^{\phi}(t') + T_{21}^{\theta}(t')) - (T_{12}^{\theta}(t) + T_{21}^{\theta}(t))
\end{aligned} \tag{5.5}$$

The number of 21-transitions in t' can be written as $T_{21}^{\phi}(t') + T_{21}^{\theta}(t')$. By Lemma 5.6.1 $\lfloor \frac{|\phi \triangle M|}{2} \rfloor$ of the $|\phi \triangle M|$ transitions in t are 12-transitions. As the first transition in t' is a 21-transition it holds that the number of 12-transitions is the same in t' and t. Because every transition that is no 12-transitions is a 21-transition it holds that

$$T_{21}^{\phi}(t') + T_{21}^{\theta}(t') = |\phi \triangle M| - \left\lfloor \frac{|\phi \triangle M|}{2} \right\rfloor \tag{5.6}$$

There is a transition at position i in t iff $i \in \phi \triangle M = (\phi \setminus M) \cup (M \setminus \phi)$, where exactly the transitions at positions in $M \setminus \phi$ are the transitions at positions in θ. Thus

$$T_{12}^{\theta}(t) + T_{21}^{\theta}(t) = |M \setminus \phi| \tag{5.7}$$

Inserting Equations (5.6) and (5.7) in Equation (5.5) completes the proof.

$$\begin{aligned}\Phi - \Theta = T_{12}^{\phi}(t) - T_{21}^{\theta}(t) &= |\phi \triangle M| - \left(|\phi \triangle M| - \left\lfloor \frac{|\phi \triangle M|}{2} \right\rfloor\right) - |M \setminus \phi| \\ &= \left\lfloor \frac{|\phi \triangle M|}{2} \right\rfloor - |M \setminus \phi| \\ &= \left\lfloor \frac{|\phi \setminus M| - |M \setminus \phi|}{2} \right\rfloor = \left\lfloor \frac{|\phi| - |M \cap \phi| - (|M| - |M \cap \phi|)}{2} \right\rfloor \\ &= \left\lfloor \frac{|\phi| - |M|}{2} \right\rfloor\end{aligned}$$

□

Analogous to Section 5.6.1 the number of binary strings with $\Phi - \Theta \geq k$ can be given based on Lemma 5.6.4.

Corollary 5.6.5. *Let (ϕ, θ) be a bipartition of $\{1, \ldots, n\}$ with $1 \in \phi$. The number of possible binary strings over $\Sigma = \{1, 2\}$ of length n, with $\Phi - \Theta \geq k$, is*

$$\sum_{i=0}^{|\phi|-2k} \binom{n}{i}.$$

Proof. Let $M \subseteq \{1, \ldots, n\}$ and (ϕ, θ) be a bipartition of $\{1, \ldots, n\}$ with $1 \in \phi$. Then by Lemma 5.6.4 $\Phi - \Theta = \left\lfloor \frac{|\phi|-|M|}{2} \right\rfloor$ holds for a binary string that is recursively defined by the set $\phi \triangle M$. As a direct consequence $\Phi - \Theta \geq k$ for all sets M with $0 \leq |M| \leq |\phi| - 2k$. Because there are $\binom{n}{|M|}$ possibilities to choose sets M of size $|M|$ the corollary follows. □

Corollary 5.6.6. *For a permutation π of length n with $\rho(\pi)$ chains the number of sorting TDRLs is given by*

$$|\mathcal{S}(\pi)| = \sum_{i=0}^{2^{\lceil \log_2(\rho(\pi)) \rceil} - \rho(\pi)} \binom{n}{i}.$$

Proof. The string t corresponding to a TDRL for π has length n. If $\Phi - \Theta \geq \rho(\pi) - 2^{\lceil \log_2(\rho(p)) \rceil - 1} = k$ holds for t then the corresponding TDRL is sorting. By substituting $|\phi(\pi)| = \rho(\pi)$ and k in Corollary 5.6.5 the result follows immediately. □

Similar to the case of restricted TDRLs it holds that there exists only one possible sorting sequence for π if $\rho(\pi)$ is a power of 2. This was not obvious in the dynamic programming approach presented in Section 5.5.2.

Also for the general case an algorithm for enumerating all sorting TDRLs is straightforward. For a given permutation π all binary strings corresponding to sorting TDRLs for π have to be enumerated. This is accomplished by enumerating all subsets $M \subseteq \{1, \ldots, n\}$ where $\phi \triangle M$

recursively defines a binary string with $\Phi - \Theta$ large enough to correspond to a sorting TDRL. The range of the size of sets M that lead to sorting TDRLs is given by Corollary 5.6.6.

5.7 Equivalence of the methods

The correctness of the results presented in the last sections implies the equality of the results. But, the correctness of the dynamic programming approach presented in Section 5.5 has not been proven formally. Hence, in this section the — not obvious — equivalence of the results of the last two sections is shown.

5.7.1 Equivalence for the restricted case

It is sufficient to show the equality of the propositions giving the number of binary strings of length n that have at least k 12-transitions. This is because the central theorems giving the number of sorting restricted TDRLs, both are direct consequences. Based on Lemma 5.6.1 an alternative proof for Lemma 5.5.1 is derived.

Alternative proof of Lemma 5.5.1: Obviously $\lfloor \frac{2k}{2} \rfloor = \lfloor \frac{2k+1}{2} \rfloor = k$ holds. By Lemma 5.6.1, each bipartition (T, E) with $|T| = 2k$ or $|T| = 2k+1$ (respectively $|E| = n - 2k$ or $|E| = n - 2k - 1$) recursively defines a binary string with k 12-transitions. Therefore, the sum of the number of possibilities to choose $2k$ (respectively $n - 2k$) elements out of a n element set plus the number of possibilities to choose $2k + 1$ (respectively $n - (2k + 1)$) elements out of a n element set is equal to the number of binary strings with k 12-transitions. That is $\binom{n}{2k} + \binom{n}{2k+1} = \binom{n+1}{2k+1}$. Respectively $\binom{n}{n-2k} + \binom{n}{n-(2k+1)} = \binom{n+1}{n-2k} = \binom{n+1}{n+1-(n-2k)} = \binom{n+1}{2k+1}$. □

The results for the number of possible binary strings of length n with at least k 12-transitions given in Corollary 5.5.2 and Corollary 5.6.2 are not obviously equal. Therefore the equality is proven formally.

Proof of the equality of Corollary 5.5.2 and Corollary 5.6.2.

$$\sum_{i=k}^{\lfloor \frac{n}{2} \rfloor} \binom{n+1}{2i+1} = \sum_{i=k}^{\lfloor \frac{n}{2} \rfloor} \left[\binom{n}{2i} + \binom{n}{2i+1} \right] \tag{5.8}$$

$$= \sum_{i=2k}^{2\lfloor \frac{n}{2} \rfloor + 1} \binom{n}{i} \tag{5.9}$$

$$= \sum_{i=2k}^{n} \binom{n}{i} = \sum_{i=0}^{n-2k} \binom{n}{i} \tag{5.10}$$

The transformation from Equation (5.8) to Equation (5.9) becomes more obvious if the sum is written as:

$$\left[\binom{n}{2k} + \binom{n}{2k+1}\right] + \left[\binom{n}{2k+2} + \binom{n}{2k+3}\right] + \ldots + \left[\binom{n}{2\left\lfloor\frac{n}{2}\right\rfloor} + \binom{n}{2\left\lfloor\frac{n}{2}\right\rfloor+1}\right].$$

Furthermore, note that the last summand $\binom{n}{n+1} = 0$ by definition.

□

The equality of the number of sorting restricted TDRLs follows immediately.

5.7.2 Equivalence for the general case

It is far from obvious that the values derived from the dynamic programming matrix (as given in Corollary 5.5.5) give the same result as the prefix sum of Pascal's triangle (Corollary 5.6.6). Therefore, in the following the equivalence is shown. For the equivalence proof the following Proposition is needed.

Proposition 5.7.1. *Let a transition string $p = p(1)\ldots p(n-1)$ be given. Let $a^x_{j,k}$, $x \in \{1,2\}$, be the number of k-sorting strings t of length j ending with x. Let Φ_j be the number of ϕ symbols in the prefix of p with length $j-1$, $1 \leq j \leq n$. Then for all j with $1 \leq j \leq n$ it holds*

$$a^x_{j,k} = \binom{j}{\Phi_j - 2k + x - 1}. \tag{5.11}$$

Proof. Proposition 5.7.1 is proven by induction over j. For $j = 1$, it holds that $\Phi_j = 0$ as the prefix of string p with length $j-1 = 0$ is empty. Therefore, $a^x_{1,k} = 1$ iff $k = 0$ and either $x = 1$ or $x = 2$. This is in agreement with the initial definition of the dynamic programming matrix. For the induction step it is assumed that Proposition 5.7.1 holds for $j < n$. In the following $a^x_{j+1,k}$ is derived. A technically necessary reformulation for the dynamic programming of Equations (5.1) and (5.2) is

$$a^x_{j+1,k} = \begin{cases} a^1_{j,k-(x-1)} + a^2_{j,k} & \text{if } p(j) = \phi \\ a^1_{j,k} + a^2_{j,k-(x-2)} & \text{if } p(j) = \theta \end{cases}$$

$a^x_{j+1,k}$ is derived for the case of $p(j) = \phi$. In this case obviously $\Phi_{j+1} = \Phi_j + 1$ holds.

$$\begin{aligned} a^x_{j+1,k} &= a^1_{j,k-(x-1)} + a^2_{j,k} \\ &= \binom{j}{\Phi_j - 2(k-(x-1))+0} + \binom{j}{\Phi_j - 2k+1} \\ &= \binom{j}{(\Phi_{j+1}-1) - 2k+2x-2} + \binom{j}{(\Phi_{j+1}-1)-2k+1} \\ &= \binom{j}{\Phi_{j+1} - 2k+2x-3} + \binom{j}{\Phi_{j+1}-2k} \\ &= \binom{j}{\Phi_{j+1} - 2k+x-1} + \binom{j}{\Phi_{j+1}-2k+x-2} \\ &= \binom{j+1}{\Phi_{j+1}-2k+x-1} \end{aligned}$$

The case of $p(j) = \theta$ (then $\Phi_{j+1} = \Phi_j$ holds) can be shown analogously:

$$\begin{aligned} a^x_{j+1,k} &= a^1_{j,k} + a^2_{j,k-(x-2)} \\ &= \binom{j}{\Phi_j - 2k+0} + \binom{j}{\Phi_j - 2(k-(x-2))+1} \\ &= \binom{j}{\Phi_{j+1}-2k} + \binom{j}{\Phi_{j+1} - 2k+2x-3} \\ &= \binom{j}{\Phi_{j+1} - 2k+x-1} + \binom{j}{\Phi_{j+1}-2k+x-2} \\ &= \binom{j+1}{\Phi_{j+1}-2k+x-1} \end{aligned}$$

Therefore, by induction Equation (5.11) holds for j, $1 \leq j \leq n$. □

As Equation (5.11) holds in particular for $j = n$, the equality is easy to proof.

Proof of equality of Corollaries 5.5.5 and 5.6.6. The sum

$$\sum_{i=\rho(\pi)-2^{\lceil \log_2(\rho(\pi))\rceil -1}}^{\lfloor \frac{\rho(\pi)}{2} \rfloor} \left(a^1_{n,i} + a^2_{n,i}\right)$$

gives the number of sorting TDRLs by Corollary 5.5.5. Using Proposition 5.7.1 and the fact that for the transition string of a permutation $\Phi_n = \rho(\pi) - 1$ holds, yields

$$\begin{aligned} \sum_{i=\rho(\pi)-2^{\lceil \log_2(\rho(\pi))\rceil -1}}^{\lfloor \frac{\rho(\pi)}{2} \rfloor} \left[\binom{n}{\rho(\pi)-2i} + \binom{n}{\rho(\pi)-(2i+1)}\right] &= \sum_{i=2\rho(\pi)-2^{\lceil \log_2(\rho(\pi))\rceil}}^{\rho(\pi)} \binom{n}{\rho(\pi)-i} \\ &= \sum_{i=0}^{\rho(\pi)-(2\rho(\pi)-2^{\lceil \log_2(\rho(\pi))\rceil})} \binom{n}{i} \\ &= \sum_{i=0}^{2^{\lceil \log_2(\rho(\pi))\rceil}-\rho(\pi)} \binom{n}{i} \end{aligned}$$

□

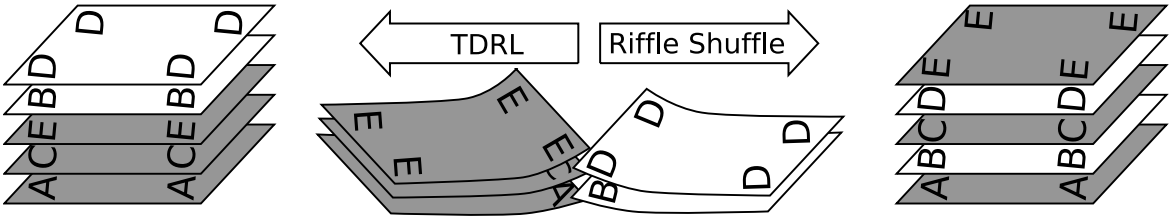


Figure 5.5 – TDRL (from right to left) and the inverse operation riffle shuffle (from left to right); reading the stacks from bottom to top gives the permutations; cards (elements) in grey correspond to cards (elements) selected in the first copy of the TDRL, respectively which are in the lower part of the stack

5.8 Recent findings

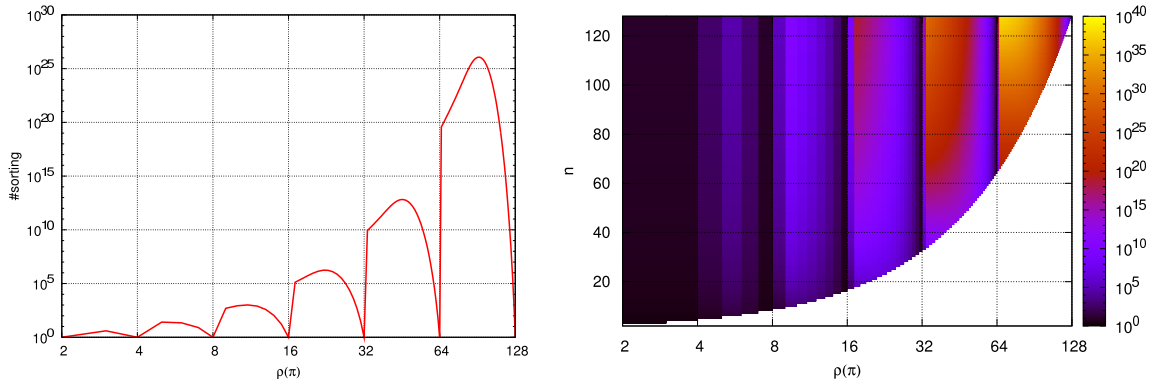
Recently, it was observed that the TDRL operation has an inverse operation. This inverse operation is the riffle shuffle as known from card playing. This operation is to cut a deck of cards into two stacks and to riffle the stacks together, i.e. interleaving the two stacks. This yields a permutation of the stack of cards. A TDRL and the corresponding riffle shuffle are depicted in Figure 5.5.

The study of the TDRL operation will benefit from this observation because the results for the riffle shuffle operation can be transferred to the TDRL operation. In (Schwenk, 1986) the question for the riffle shuffle was raised. The answer was given in (Schwenk, 1988) in perfect agreement with the TDRL distance presented in Chaudhuri et al. (2006). The statistics and the study of the significance of TDRLs profit from available work on the statistics of random riffle shuffles (Aldous and Diaconis, 1986; Bayer and Diaconis, 1992). Most important for this work is (Grinstead and Snell, 2006, Chapter 3.3) because closed formulas for the number of sorting riffle shuffle scenarios are given, i.e. the number of sorting TDRL scenarios.

5.9 Results

In this section the relevance of the theoretical findings for the analysis of mitochondrial gene orders is shown. Moreover, random TDRLs are applied to the identity permutation and present the number of resulting chains. The result is then used to support scenarios of a sequence of TDRLs. Such a very likely scenario is presented for mitogenomes.

Figure 5.6a gives the number of sorting restricted TDRLs for permutations having $\rho(\pi) \in [2 : 128]$ chains. Recall, the number of sorting restricted TDRLs does only depend on the number of chains and not on the length of the permutations. The figure shows that the number of sorting restricted TDRLs $|\mathcal{S}_r|$ is 1 for all cases where $\rho(\pi)$ is a power of 2, but can be very large otherwise. $\mathcal{S}_r$ is smaller the closer $\rho(\pi)$ gets to the next smaller or larger power



(a) **(b)**

Figure 5.6 – a) Number of sorting restricted TDRLs for different numbers of chains $\rho(\pi)$; b) Number of sorting TDRLs for permutations of different length n and different numbers of chains $\rho(\pi)$; note that the number of chains is plotted with a $\log_2$ scale and the number of (restricted) sorting TDRLs with a $\log_{10}$ scale

of 2. For example the maximum for $|\mathcal{S}_r|$ is more than 10^{26} for $\rho(\pi) = 92$, towards 128 and 64 the number of sorting TDRLs decreases. In Figure 5.6b the number of sorting TDRLs is given for permutations of length $n \in [2 : 128]$ having $\rho(\pi) \in [2 : 128]$ chains. Recall, the number of sorting TDRLs depends on n in the general case. While for some values of $\rho(\pi)$ the number of TDRLs is immense — e.g. more than 80 million for $n = 37$ (the length of mitochondrial gene orders) and $\rho(\pi) = 9$ — it is 1 if the number of chains is a power of 2. This fact can be used to identify sequences of TDRLs for which no alternative sorting scenario exist.

Figure 5.7 shows the results for the case that $r \in \{2, 3, 4, 5, 6, 8, 12, 16\}$ random TDRLs have been applied to the identity permutation of length 37 resulting in permutation π. For each r this has been repeated 10 000 times resulting in a set of permutations Π_r. Two models are used for the generation of random TDRLs:

i) the complete permutation is duplicated in tandem and
ii) only a randomly chosen interval (start and end point are chosen at random) is duplicated.

For both methods it is decided at random if a duplicated element is kept in the first or in the second copy. The histograms for the number of resulting chains for Π_r are depicted in Figure 5.7a for random TDRLs applied to complete permutations and Figure 5.7b for random TDRLs applied to intervals that are chosen at random. Note, the model where TDRLs only affect random intervals is intuitively biologically more interesting. Additionally the case when the permutations are chosen at random is shown. The case $r = 1$ is omitted because it leads almost certainly to a permutation with two chains. This is easy to see because there are only $l + 1$ TDRLs without effect, where l is the number of elements in the tandem duplicated interval, i.e. resulting in ι which has only one chain, and the remaining out of the 2^l possible

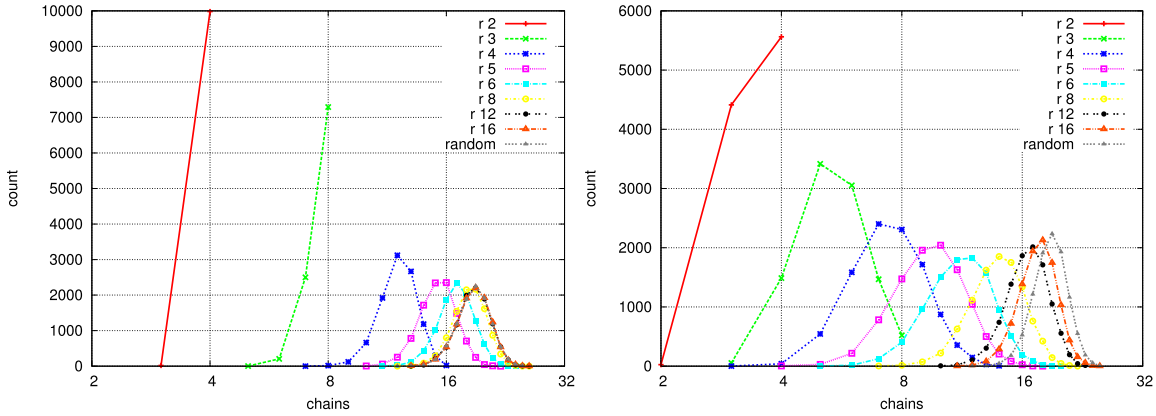


Figure 5.7 – Number of chains of 10 000 permutations Π_r of size 37 for each $r \in \{2, 3, 4, 5, 6, 8, 12, 16\}$; r is the number of TDRLs applied to the identity permutation to obtain a permutation $\pi \in \Pi_r$; a) random TDRLs applied to the complete permutation; b) random TDRLs applied to intervals that are chosen at random; for the line denoted with random each π is a random permutation

TDRLs lead to a permutation with two chains. Thus, a permutation with two chains is almost certainly the result of a single TDRL. Similarly, a permutation with four chains is the result of two TDRLs with a high probability.

In the random TDRL model where random intervals are duplicated in 7 092 cases a permutation with four chain was generated. In the great majority (5 562 cases) this was due to four TDRLs (1 487 cases due to three TDRLs, 39 cases due to four TDRLs, and four cases due to five TDRLs). Thus, a permutation with four chains was generated with high probability by two TDRLs, assuming this model for the random TDRLs.

In the random TDRL model where the complete permutation is duplicated in 9 981 cases a permutation with four chains was generated and in all those cases this was a result of two random TDRLs. In only 19 permutations generated with two random TDRL three chains are found. Hence, in this model of random TDRLs, it is nearly certain that a permutation with four chains was generated by two TDRLs. For eight chains, there is a very high probability that this is due to three TDRLs. There are 7 307 cases where a permutation with eight chains was found. In 7 293 cases this was due to three TDRLs and in 14 cases this was due to four TDRLs. The remaining 2 707 permutations generated with three TDRL have less than eight chains.

For larger number of chains, i.e. 16 and more chains in the case of random TDRLs applied to complete permutations, respectively eight and more in the case where only a random interval in duplicated by the random TDRLs, it is not possible to decide with high probability by how many random TDRLs they have been created. This is because with increasing r the number

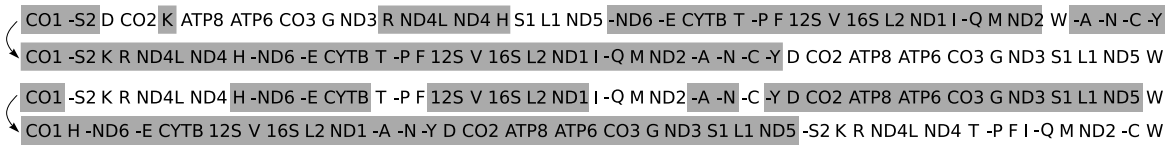


Figure 5.8 – The unique TDRL scenario from *Salvelinus fontinalis* (top row) to *Porichthys myriaster* (bottom row); genes kept in the first copy are boxed

of chains found in the generated permutations approaches the number of chains expected in random permutations.

In order to show the potential of the presented theoretical results all pairs of existing complete mitochondrial gene orders from the NCBI database have been analysed and a sequence of TDRLs has been manually identified that can be considered as biologically likely. This example is the sequence of two TDRLs transforming the gene order of *Salvelinus fontinalis* (which has the typical vertebrate gene order) into *Porichthys myriaster*, given in Section 5.9. The TDRL scenario is supported by the following observations. Scenarios considering other rearrangements are much longer, e.g. the inversion distance is 15 and the transposition distance is seven. The transposition distance was computed with an exact algorithm provided by M. Bader. Note, the lower bound of the distance is also seven (Bafna and Pevzner, 1995). The given scenario is the only parsimonious sorting scenario based on TDRLs. That is i) the TDRL scenario in the given direction is unique, ii) the TDRL distance in the other direction of length 3 is not parsimonious, and iii) there exists no ancestral permutation of *S. fontinalis* and *P. myriaster*, such that the two genomes can be reached with two or less TDRLs (this was checked by a computationally expensive brute force algorithm). Due to the histograms as presented in Figure 5.7, it is clear that the four chains indeed occurred very likely due to two TDRLs. Furthermore, according to Miya et al. (2005) duplications and losses are supported by fragments of nucleotide sequences in the mitogenomes.

However, the presented results for the pair of mitochondrial gene orders are not final. The main open question, from a theoretical point of view, is whether the presented results can be applied to circular gene orders or modification are necessary. Furthermore, the accuracy of the data set is not guaranteed — even if the data set was considerably improved — and a manual analysis is inevitable. Of course all conclusions can only be drawn on the basis of the currently available data.

One example for the concerns with the data set is presented in the following. Due to the improvements of the data sets (Appendix A) it was observed that the gene order of *C. sloani* can also be transformed with two TDRLs into the gene order of *P. myriaster*. These two TDRLs are depicted in Section 5.9. Note, the properties of the TDRL scenario are identical to the properties of the TDRL scenario for *S. fontinalis* and *P. myriaster*. That is the necessary

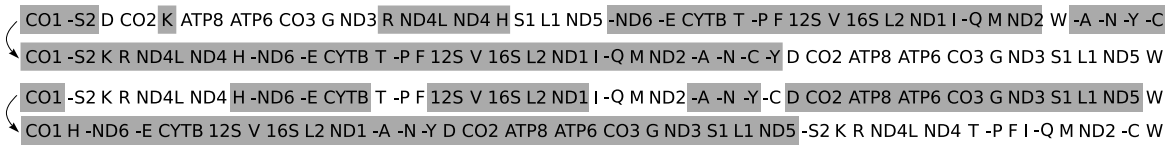


Figure 5.9 – The unique TDRL scenario from *Chauliodus sloani* (top row) to *Porichthys myriaster* (bottom row); genes kept in the first copy are boxed

number of alternative inversions and transpositions and the absence of a TDRL median. The only difference between the gene orders of *C. sloani* and *S. fontinalis* is the swap of tRNAs C and Y. This is reflected in the TDRL scenario by a difference in the second TDRL where the assignment of the tRNAs C and Y to the first or second copy is interchanged in the two scenarios.

Summarising, there are strong indications for the presented TDRL scenario to *P. myriaster*. But there remains uncertainty of how the tRNAs C and Y have been rearranged. Note, the genomic rearrangement events for the gene orders of this example have not been described in the literature before.

5.10 Conclusion

Tandem duplication random loss (TDRL) events are important gene order rearrangement operations especially in mitochondrial gene orders. In this chapter combinatorial properties of the TDRL operation have been studied. The set of all sorting TDRLs has been analysed, i.e. the set of TDRLs that cause a permutation to become closer to another given permutation. Additionally an interesting restricted case of the problem has been investigated which leads to an analysis of the general case. Methods for the enumeration of all sorting (restricted) TDRLs and closed formulas for the number of sorting (restricted) TDRLs have been presented. Algorithms for enumerating all sorting TDRLs have been obtained by an enumeration of binary strings with certain properties. The presented theoretical results significantly extend previous works, by providing an alternative method for the enumeration and counting problem. With this approach a combinatorial interpretation for the closed formulas of the general case could be given.

The relevance of the theoretical findings when identifying sequences of TDRLs for real biological data, e.g. mitochondrial gene orders has been shown.

6 Conclusion

Two fundamental genome rearrangement problems have been studied in this work. These are the sorting problem and the inversion median problem. Both problems have been considered with modified rearrangement models in order to obtain more plausible solutions. It was shown that increased plausibility can be accompanied by an efficient solution.

The first problem analysed in this work is the inversion median problem (IMP). This is to find a median gene order minimising the inversion distances to given gene orders. This problem was studied in the modified rearrangement model allowing no inversion in the median scenario to break one of the common intervals of the input permutations. This is the preserving inversion median problem (pIMP). Three exact algorithms for the preserving inversion median problem have been proposed. Algorithms CIP and ECIP are based on Caprara's branch and bound median solver for the IMP. Both algorithms modify the branch and bound search. While CIP checks if a generated solution is preserving, ECIP restricts the branch and bound search such that only preserving solutions are generated. The algorithm TCIP for solving the preserving inversion median problem (pIMP) for k given gene orders is based on an extension of strong interval trees, namely k-signed strong interval trees. The relation of the difficulty of the pIMP and the structure of the corresponding k-signed strong interval tree was analysed theoretically. Based on these results, algorithm TCIP was designed such that it can solve a pIMP instance by solving several smaller instances of the IMP. For computing one preserving median instead of all, several non-trivial techniques are applied within algorithm TCIP to increase efficiency.

Properties of the preserving inversion median problem and the three presented algorithms have been studied empirically for simulated data sets, gene orders of mitochondrial genomes of *Metazoa*, and chloroplast gene orders of *Campanulaceae*. It has been shown empirically for the simulated and biological data sets that common intervals occur often and many common intervals are destroyed by solutions for the IMP, i.e. when common intervals are not considered. It was demonstrated that the algorithms CIP and ECIP can solve the pIMP for most of the simulated and biological data sets in reasonable time. Properties of the corresponding strong interval trees, which are of importance for the run time behaviour

of TCIP, have been analysed for data sets of mitochondrial gene orders and for randomly generated data sets. The empirical study has demonstrated that often TCIP has to solve no IMP instances for solving the pIMP. This leads to a linear run time behaviour of TCIP. Recall, the preserving inversion median problem is NP-hard. Furthermore, only a few and smaller instances of the IMP have to be solved in the remaining instances . The empirical study has clearly shown that TCIP outperforms CIP and ECIP by orders of magnitude. Furthermore, preserving median scenarios can be computed with TCIP even faster, than standard (i.e. not necessarily preserving) median scenarios can be computed with the state of the art IMP solver of Caprara (and a variant thereof). This is interesting because even computing the preserving minimum inversion distance between two gene orders is an NP-hard problem (whereas the same problem with not necessarily preserving inversions can be solved in polynomial time). Furthermore, it was demonstrated that algorithm TCIP can be used as a good heuristic for the IMP.

The sorting problem was studied in the second part of the work. The heuristic algorithm `CREx` for computing rearrangement scenarios for pairs of given unichromosomal gene orders was presented. The rearrangement operations inversion, transposition, inverse transposition, and tandem duplication random loss are considered by `CREx`. This covers the biologically evident operations for *Metazoan* mitochondrial gene orders. `CREx` reconstructs a rearrangement scenario by identifying patterns in signed strong interval trees which represent the common intervals of gene order pairs. Thus, `CREx` realises both types of modified rearrangement models studied in this thesis. Several extensions of `CREx` for the handling of alternative scenarios, ordered scenarios, and combinations of inversions and tandem duplication loss events have been described. In a large empirical study the quality of the `CREx` reconstructions was analysed for simulated data sets generated with several rearrangement models. Parameters of the simulation properties of the strong interval trees have been identified where `CREx` returns results of high quality.

Based on `CREx`, further gene order rearrangement methods for the reconstruction of phylogeny have been presented. The algorithm `TreeREx` utilises the pairwise scenarios computed by `CREx` to infer ancestral permutations and genomic rearrangement operations in a given binary phylogenetic tree. `TreeREx` was applied to biological data sets of mitochondrial gene orders of *Echinodermata* and *Teleostei*. In both data sets the reconstructed rearrangements are in strong correspondence with published results. Based on the results of the `CREx` for simulated data, a new simple method was introduced to explore the rearrangements of a data set without a given tree. The method was applied to the complete data set of *Metazoan* mitochondrial genomes. The method obtains very efficiently the complete picture of the rearrangements within the *Metazoan* phyla. The returned results are compliant with the literature to a large extent. The presented algorithms `CREx`, `TreeREx`, as well as the new exploration method solved the simulated and biological data sets very efficiently.

Tandem duplication random loss (TDRL) events are important gene order rearrangement operations especially in mitochondrial gene orders. A better understanding of this operation is indispensable for the study of mitochondrial gene orders. Thus, combinatorial properties of the tandem duplication random loss rearrangement have been studied in the last chapter of this work. The set of all sorting TDRLs and an interesting restricted case of the problem has been investigated. Methods for the enumeration of the set of sorting (restricted) TDRLs and closed formulas for calculating the number of sorting (restricted) TDRLs have been presented. The results are obtained by an enumeration of binary strings with certain properties. The relevance of the theoretical findings when identifying sequences of TDRLs for real biological data, e.g. mitochondrial gene orders, has been shown.

A Mitochondrial gene order data set

The most up-to-date source for mitochondrial genomes and their annotation, i.e. information on the genes that are found in the genome (name, position, strand, etc.), is NCBI RefSeq (Pruitt et al., 2007). The RefSeq data base grew rapidly since the first mitochondrial genomes have been submitted in the early 1990's (see Figure A.1). Despite the efforts to improve the quality of the data, the annotations are highly inconsistent. That is there is no standard for naming the genes, additionally the annotations contain a large number of errors (Boore et al., 2005; Perseke et al., 2008). These errors are for example: i) genes are annotated on the reverse complement strand, ii) genes are annotated with a wrong name (most error prone are the pairs of tRNAs L1, L2 and S1, S2 which are distinguished by their anticodon), iii) missing genes, iv) genes that are annotated too often, and v) wrong start and end positions. All these sources of error may mislead gene arrangement analyses.

There are manually improved databases for mitochondrial genomes (Boore's mitochondrial gene arrangement guide (Boore, 2001), OGRe (Jameson et al., 2003), and AMIGA (Feijão et al., 2006)). Boore's gene arrangement guide has been the standard source for mitochondrial gene arrangements for a long time, but unfortunately it is not available anymore and therefore outdated. AMIGA covers only the *Arthropod* part of the *Metazoan*. Furthermore, it is not possible to obtain the gene orders of genomes from the web site. Being not standardised and therefore not reproducible is one possible problem of manual improvements. Additionally, the growing amount of data is hard to manage manually. Another approach to

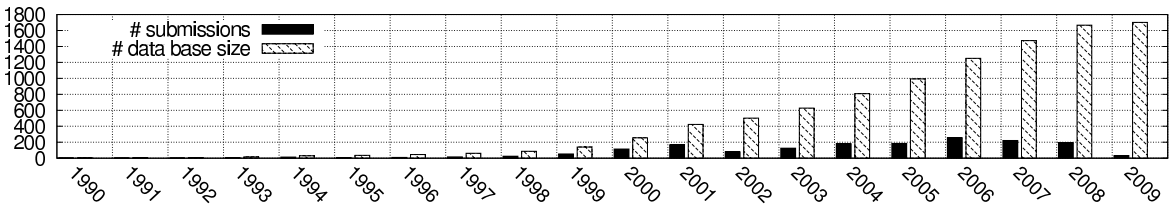


Figure A.1 – Number of *Metazoan* mitochondrial genomes newly submitted to (respectively included in) the RefSeq data base from 1990-2009

correct these errors is a re-annotation of the genomes. DOGMA (Wyman et al., 2004) is an automated method for the re-annotation of mitochondrial genomes. This program can not be used here because it needs manual interaction and is not suitable for large scale re-annotation in its current implementation as a website.

For these reasons a project in cooperation with the Bioinformatics Department of the University of Leipzig aiming for the automatic re-annotation of the *Metazoan* mitochondrial genomes was started. This project began with the analysis of an Echinoderm data set (Perseke et al., 2008) and is still ongoing. With the current state of the project the re-annotation of mitochondrial genomes is not yet possible, but finished components of the project can be used to improve the RefSeq data and derive a high quality gene arrangement data set. The construction of the data set is described in the following, but first note that it is not possible to correct errors in the sequences of the genomes, incomplete, or wrong assembled sequences. Mistakes in the sequence affect the annotation and consequently also the arrangements. These errors are rarely reported because re-sequencing is rare, a case where the sequence mislead an arrangement analysis is reported in Ren et al. (2009).

The improvement of the annotation focuses on the 22 tRNA genes that are expected in a mitochondrial genome because the annotation of the tRNA genes is most error prone. The programs tRNAscan-SE version 1.23 (Lowe and Eddy, 1997) and ARWEN version 1.2.3 (Laslett and Canbäck, 2008) were used for the improvement. These are programs to predict tRNAs in genomic sequences.

The mitochondrial gene order data set is based on NCBI RefSeq (Pruitt et al., 2007) Release 36 from July 13, 2009[1]. This release contains 1 701 complete reference assemblies of mitochondrial genomes of *Metazoan* species. A self developed parser was used to extract the annotation and remove the inconsistent naming of the genes. The extracted annotation was post processed in the following way.

1. From the ARWEN and tRNAscan-SE predictions all tRNAs are discarded which i) do not have a standard name (i.e. where the programs failed to determine the tRNA type) and ii) tRNAs which overlap by more than 50% of the length with an annotated protein or rRNA are discarded

2. Renaming: For each tRNA in the GenBank annotation the corresponding predictions of ARWEN are determined. This is done by finding the ARWEN prediction which overlaps most with the GenBank annotation of the tRNA. Where at least an overlap of 70% is required. If the determined pair of tRNAs has different names, this is stored as a candidate for renaming. The tRNAscan-SE predictions are preprocessed in the same manner. From the determined rename candidate pairs, all pairs are discarded if there

[1] `ftp://ftp.ncbi.nih.gov/refseq/release/mitochondrion/mitochondrion1.genomic.gbff.gz`

is disagreement in the renaming between the tRNAscan-SE and ARWEN predictions. Unspecified GenBank annotation S or L are renamed to the name of the tRNAscan-SE or ARWEN prediction. Let $R = \{(a_1, p_1), \ldots, (a_r, p_r)\}$ be the remaining set of r rename pairs where a_i are the names of the GenBank annotation and p_i are the names of the ARWEN or tRNAscan predictions, with $i \in [1 : r]$. "Rename cycles" are determined from R, i.e. sequences $(x_1, x_2, \ldots, x_s)$ with $(x_s, x_1) \in R$ and $(x_i, x_{i+1}) \in R$ for $i \in [1 : s - 1]$. All re-namings, implied by the "rename cycles", are applied.

3. Multiplicity: If some of the 22 tRNAs, expected to be found in mitochondrial genomes are missing in the GenBank annotation, the gap is filled with the ARWEN or tRNAscan-SE prediction as long as a) tRNAscan-SE and ARWEN predict at most one tRNA of the type, b) they do not disagree, i.e. the predictions must be at the same position (overlap of more that 70%), and c) the predictions must have the same name.

   In case of finding supernumerous tRNA in the GenBank annotation, the following treatment is applied. If all but one of the duplicates of a tRNA do not overlap by at least 70% with a tRNAscan-SE or ARWEN prediction, all those duplicates are removed and only the single tRNA, which is also found by tRNAscan-SE or ARWEN, is kept.

4. Strand: Again, the tRNAscan-SE and ARWEN prediction with a maximal overlap with the GenBank annotation is determined. The overlap must also be at least 70% here. The strand of the GenBank annotation is changed if i) the names of the GenBank annotation and the maximal overlapping tRNAscan-SE and ARWEN predictions have no disagreement, ii) the annotation differs from the tRNAscan-SE or ARWEN prediction, and iii) there is no disagreement on the strandedness of the tRNA in the tRNAscan-SE and ARWEN predictions.

For the construction of the gene arrangement data set, all gene orders have been discarded which do not have the set of 37 genes common to almost all mitochondrial gene orders, i.e. i) 13 protein coding genes (CO1, CO2, CO3, NAD1, NAD2, NAD3, NAD4, NAD4L, NAD5, NAD6, ATP6, ATP8, and CYTB) ii) two rRNA genes (16S and 12S), and iii) 22 tRNA genes A, C, D, E, F, G, H, I, K, L1, L2, M, N, P, Q, R, S1, S2, T, V, Y, W (Boore, 1999). For *Nematodes* and *Platyhelminthes* a set of genes, reduced by ATP8 was used because almost all species of these groups do not have this gene.

For the data set of all *Metazoan* mitochondrial genomes, excluding the *Nematodes* and *Platyhelminthes*, the following results are obtained. The renaming procedure as described above corrected the name of 203 tRNAs in 73 species. The names of the Leucine and Serine tRNAs have been corrected mostly. The names of L1 and L2 (respectively S1 and S2) have been swapped twelve (32) times, in 50 cases unspecified L tRNAs have been renamed to L1 or L2 (each 25 times), in 30 (29) cases unspecified S tRNAs have been renamed to S1 (S2),

and in three cases other tRNAs names have been swapped (E and Q, P and T, S2 and W). With regard to erroneous tRNA multiplicity the approach was successful, too and corrected the multiplicity of 166 tRNAs in 107 species. Missing genes could be identified in 108 cases and up to four duplicate tRNAs have been removed from the annotations in 58 cases. The duplicity of some tRNAs was corrected more often. S1 was missing in 19 species, and I was annotated twice in 20 species. For all other tRNAs the duplicity was corrected in less than ten species. The most changes have been done with respect to the strandedness of the tRNAs. The strand was corrected in 327 tRNAs in 195 species. In the majority of 290 cases the strand was changed from + to −. This is not surprising as annotating a gene on the negative strand requires to write an additional information in the GenBank (the string "complement" has to be added), whereas the annotation to the positive strand does not require additional information. This indicates that this additional information is often forgotten. Regarding the strandedness, some tRNAs have also been corrected more often: A 16x, C 25x, E 36x, N 22x, P 59x, Q 47x, S2 36x, and Y 32x. Except of four S2 tRNAs all these tRNAs have been changed from the + to the - strand. All other tRNAs have been corrected less than seven times.

When comparing the gene arrangements derived from the original and the improved RefSeq data set, the following observations can be made. 1 404 gene arrangements are derived from the original data without *Nematodes* and *Platyhelminthes*, 239 are excluded because of non standard gene content). Due to the applied improvements, 54 more species are included in the data set. The data set contains many duplicated gene arrangements. The original data includes 247 and the improved data set 185 unique gene orders. This indicates that the errors in the RefSeq annotations artificially increase the number of distinct gene arrangements. Thus, lots of artificial rearrangements can be found (single gene inversions in the case of wrong strandedness, and more dramatic rearrangements when gene names are switched) in the gene arrangement data set derived from the original data.

A few differences can be found between the original and the improved data set of *Nematodes* and *Platyhelminthes*. The strandedness is corrected in three cases. The names of six pairs of tRNAs are swapped (four times L1/L2 and two time S1/S2), one unspecified L was renamed to L2. The multiplicity of tRNAs was corrected in two species (six missing tRNAs and one duplicated). 49 gene orders are implied from the original (13 are not included because of non standard gene content). These 49 gene orders include 21 unique gene orders. Due to the applied improvement strategy, two more gene orders are included in the data set. The number of unique gene orders decreases also for this data set, in this case to 18.

The former standard data base for mitochondrial gene arrangements has been Boore's mitochondrial gene arrangement guide (Boore, 2001) respectively the more up to date website at (Boore, 2006b). In the following, a comparison of the data set from (Boore, 2006b) from December 2006 and the improved data set is given. 875 of the gene orders of the improved

RefSeq data set are not present in Boore's gene orders, and nine gene orders of Boore's data set are not present in the RefSeq data set. From the remaining gene orders, which are present in both data sets, the great majority of 515 gene orders are equal and only 68 not. This shows already that the obtained data set has a high quality. A closer look on the differences reveals that most of the differences are due to a difference of the strandedness of single genes, i.e. the strand of 117 genes differed in 62 species. Swapping the names of S1 and S2 explains the differences in three more species. The remaining three differences are due to changes in the position of a single tRNA. For the *Nematodes* and *Platyhelminthes*, all gene orders are equal when comparing the 18 gene orders in common (33 are not in Boore's data base).

The improved gene order data set was also compared to the gene orders from the OGRe data base which contains 170 unique gene orders. The majority of 110 gene orders has shown no difference to the improved data set. The differences are mainly in the strandedness of single tRNAs, i.e. the strand of 121 tRNAs differs. In nine cases genes are renamed respectively two adjacent genes changed the position. In three cases a tRNA is found at another not adjacent position.

The presented statistics and comparisons show that the applied improvement strategy leads to a data set of higher quality.

The improved mitochondrial data set containing the gene orders of all *Metazoan* species, which have the standard set of 37 genes, was subdivided in order to allow the analysis of (smaller) subgroups of interest. This was done in a phylogenetic meaningful way such that the resulting groups have similar sizes. The considered groups are:

1. *Chordata* and the subgroups *Actinopterygii* and all non *Actinopterygii* species (including *Amphibia*, *Archosauria*, *Mammalia*, and others)

2. *Arthropoda* (two gene orders from *Priapulida*, *Onychophora* are included in this group) and the four subgroups *Chelicerata*, *Crustacea*, *Hexapoda*, and *Myriapoda*

3. Furthermore, *Echinodermata*, *Hemichordata*, and *Xenoturbellida* are grouped together as basal *Deuterostomia*

4. the *Lophotrochozoan* groups *Annelida, Brachiopoda, Echiura, Mollusca, Sipuncula*, and *Entoprocta*

The number of gene orders, the number of unique gene orders, and the error statistics of the considered data sets are summarised in Table A.1.

group		error statistics			# gene orders	
name	abrv	ren	mul	sig	tot	uni
• All (standard gene set)	(all)	203	166	327	1 458	185
• *Arthropoda*	(art)	30	34	37	234	77
◦ *Chelicerata* and *Myriapoda*	(cmy)	3	6	12	45	27
◦ *Crustacea*	(cru)	6	1	12	41	20
◦ *Hexapoda*	(hex)	21	25	13	146	30
• *Chordata*	(cho)	157	24	268	1 155	61
◦ *Actinopterygii*	(act)	59	13	105	555	28
◦ non *Actinopterygii*	(nac)	98	11	163	600	39
• *Echinodermata*, *Hemichordata*, and *Xenoturbellida*	(ehx)	0	10	9	20	11
• *Annelida, Brachiopoda*, *Echiura, Mollusca*, and *Sipuncula*	(abm)	10	37	13	34	23
• *Nematodes*, *Platyhelminthes*	(np)	13	7	3	51	18

Table A.1 – Statistics for the improvement of the *Metazoan* gene order data set and size of the data set for the different subgroups; ren: number of renamed genes; mul: number of corrected gene multiplicity problems; sig: number of corrected strandedness errors; tot: total number of gene orders with the standard set of genes; uni: number of unique gene orders in the data set

B The *chordate* connected component and the rearrangements

Figure B.1 depicts the connected component consisting of *Chordata*, one *Hemichordata*, and one *Xenoturbellida*. The rearrangements of this component are not shown here.

The rearrangement scenarios from the connected components of the graph presented in Section 4.5 are given in the following. Each of the Figures B.2–B.19 shows the rearrangement scenarios for one of the connected components from Figures 4.17–4.22. The rearrangement scenarios of the huge *arthropod* component, shown in Figures 4.22 and 4.23, are depicted in Figures B.20–B.26. Different rearrangement scenarios are separated by a horizontal line.

In order to save space, each rearrangement scenario is given as follows. Let π and σ be two gene orders and $\rho_1, \ldots, \rho_k$ be a rearrangement scenario from π to σ. The rearrangement scenario is given in k lines where line i, with $1 \leq i \leq k$, present the strong interval tree of the two permutations rearranged by the first $i-1$ rearrangements of the scenario. That is reading the leaves of the tree from left to right gives the permutation after the first $i-1$ rearrangements are applied. In the strong interval tree of the i-th line the rearrangement ρ_i is marked (the colours are as is Figure 4.1). The result of the last rearrangement (i.e. σ) is not shown. The unique identifier of the rearrangement is given on the left.

Rearrangements which are compliant to the literature are marked with •. Differently reconstructed rearrangements or rearrangements that may be caused by annotation errors are marked with •. Rearrangements that have not been found in the literature are marked with °.

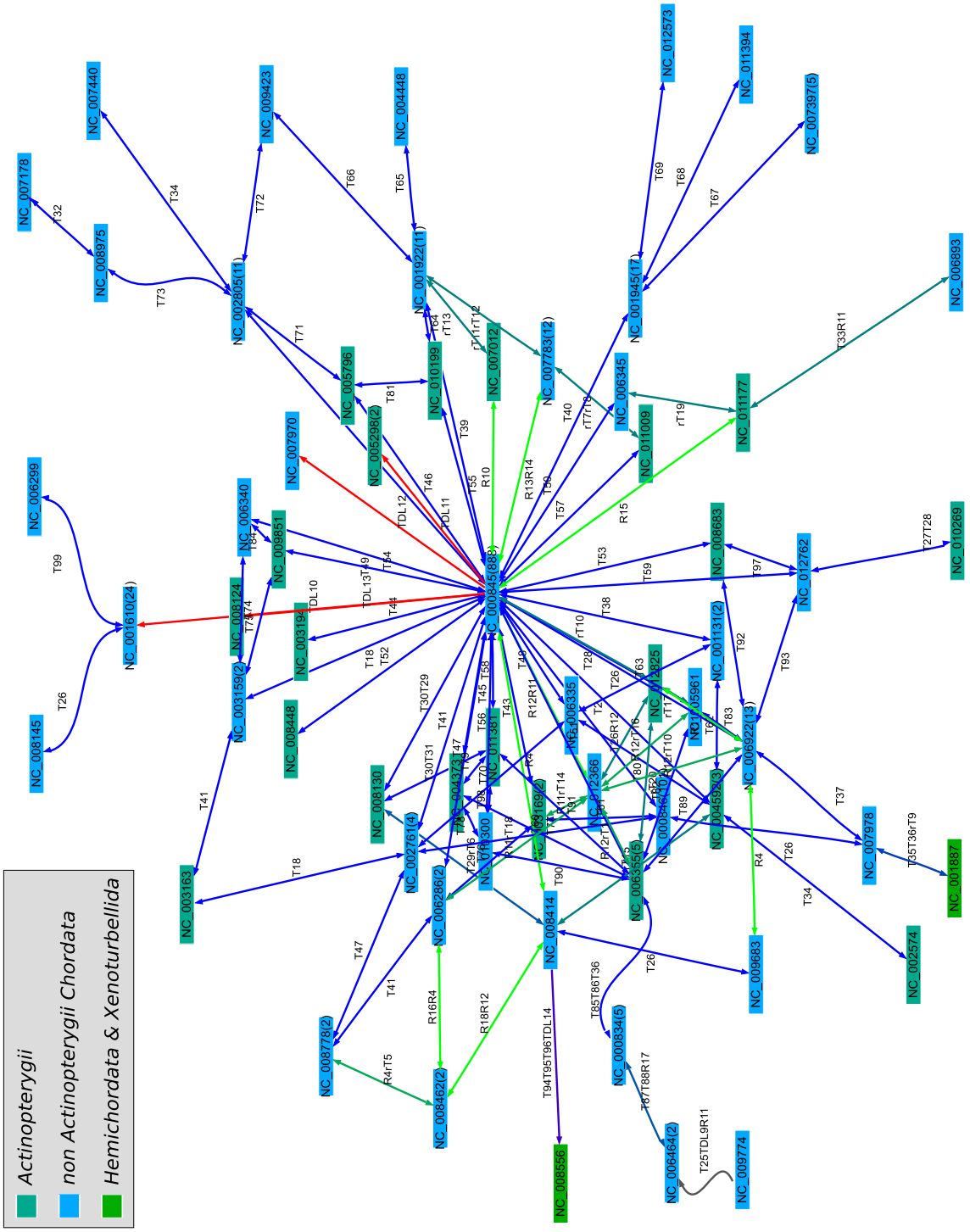


Figure B.1 – The largest connected component including the mitochondrial gene orders from *Chordates* (blue), one *Hemichordata*, and *X. bocki*

Echinodermata

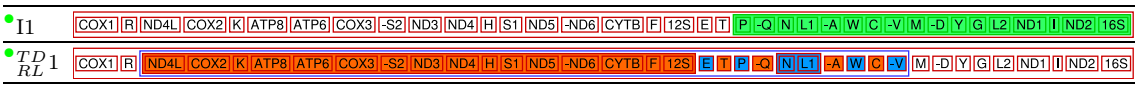


Figure B.2 – Rearrangements from the connected component shown in Figure 4.17a; from top to bottom: NC_001453-NC_001627, NC_001453-NC_005929

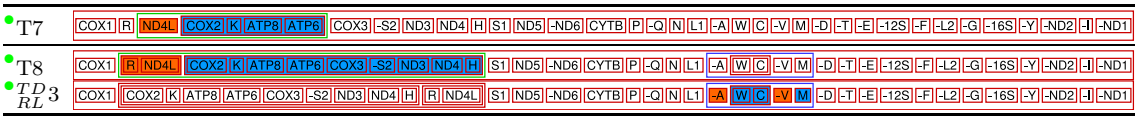


Figure B.3 – Rearrangement scenarios from the connected component shown in Figure 4.17b; from top to bottom: NC_001878-NC_007689, NC_001878-NC_010692

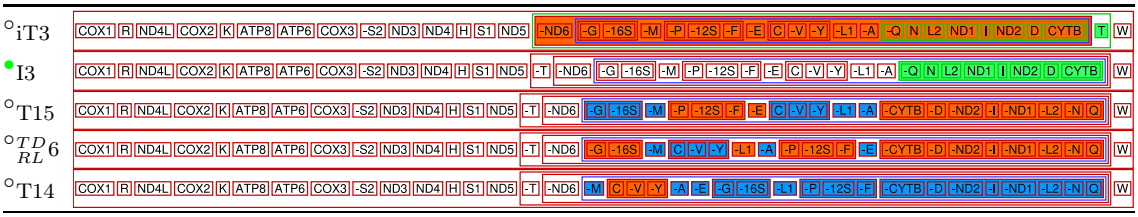


Figure B.4 – Rearrangement scenario from the connected component shown in Figure 4.17c, i.e. NC_005930-NC_005334

Mollusca and *Annelida*

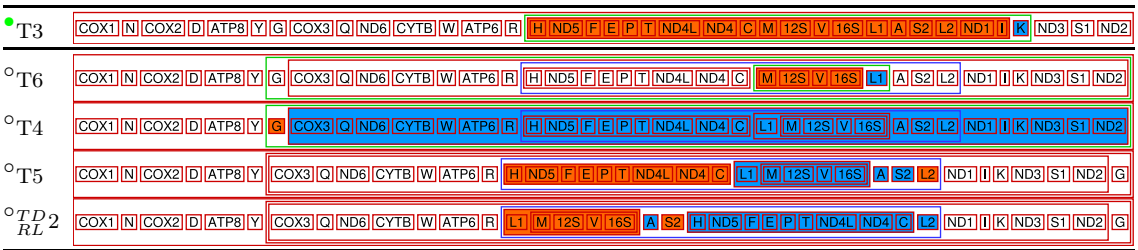


Figure B.5 – Rearrangement scenarios from the connected component shown in Figure 4.18a; from top to bottom: NC_001673-NC_006321 and NC_001673-NC_007933

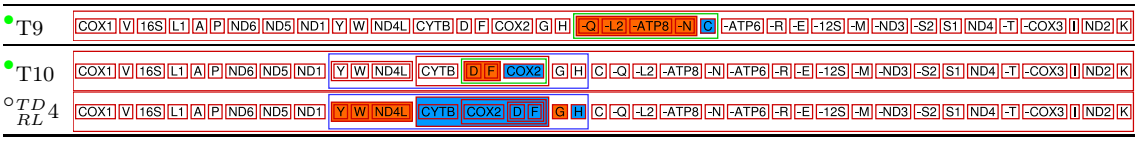


Figure B.6 – Rearrangement scenarios from the connected component shown in Figure 4.18b; from top to bottom: NC_002176-NC_004321, NC_004321-NC_012383

Figure B.7 – The transposition from the connected component shown in Figure 4.18c; NC_002507-NC_007894

Figure B.8 – TDRL from the connected component shown in Figure 4.18d; NC_005335-NC_011763

Figure B.9 – Rearrangement scenarios from the connected component shown in Figure 4.19a; from top to bottom: NC_001816-NC_005439, NC_001761-NC_001816, NC_001761-NC_005439, NC_005439-NC_010220, and NC_005439-NC_012434

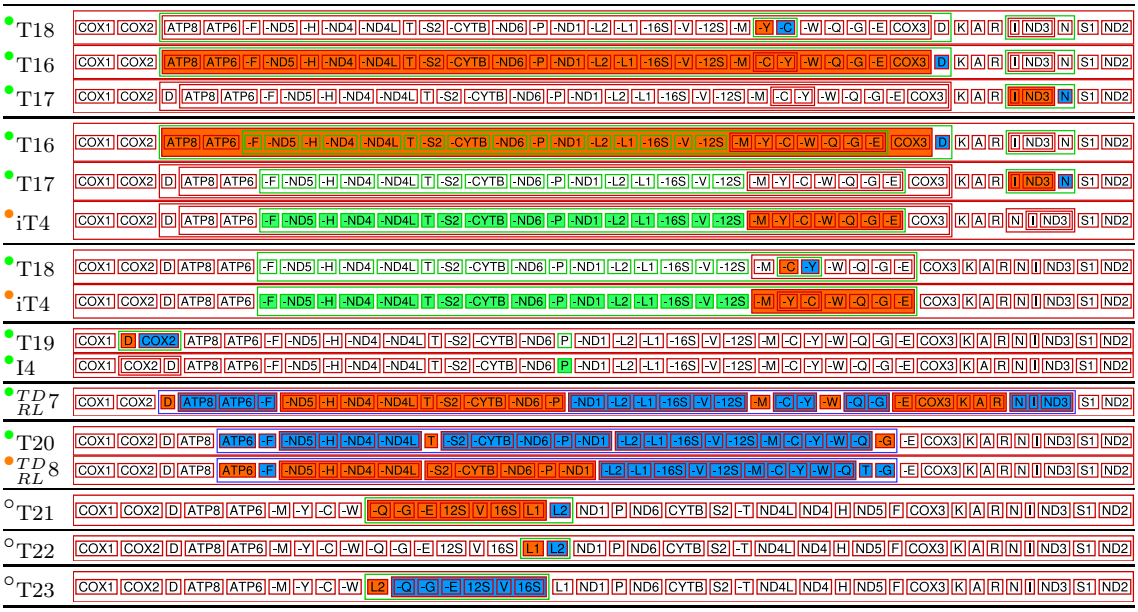


Figure B.10 – Rearrangement scenarios from the connected component shown in Figure 4.19b; from top to bottom: NC_005940-NC_006353, NC_005940-NC_007781, NC_006353-NC_007781, NC_001636-NC_006353, NC_006353-NC_007895, NC_006353-NC_007980, NC_007781-NC_008797, NC_007781-NC_012899, and NC_008797-NC_012899

Arthropoda

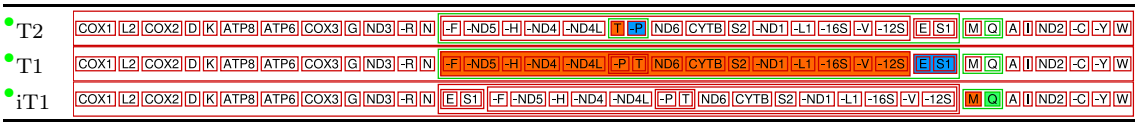


Figure B.11 – Rearrangement scenario from the connected component shown in Figure 4.20a; NC_001566-NC_010967

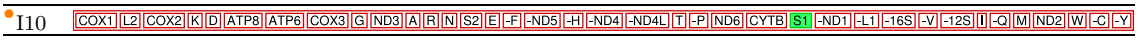


Figure B.12 – Inversion from the connected component shown in Figure 4.20b; NC_008756-NC_009770

Figure B.13 – Transposition from the connected component shown in Figure 4.20c; NC_003979-NC_008831

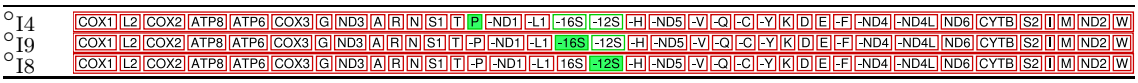


Figure B.14 – Rearrangement scenario from the connected component shown in Figure 4.20d; NC_006992-NC_011597

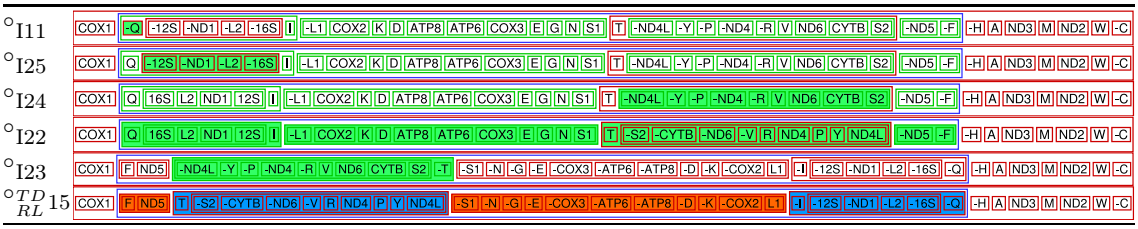


Figure B.15 – Rearrangement scenario from the connected component shown in Figure 4.20e; NC_010595-NC_010596

Figure B.16 – Inversion from the connected component shown in Figure 4.20f; NC_010526-NC_012571

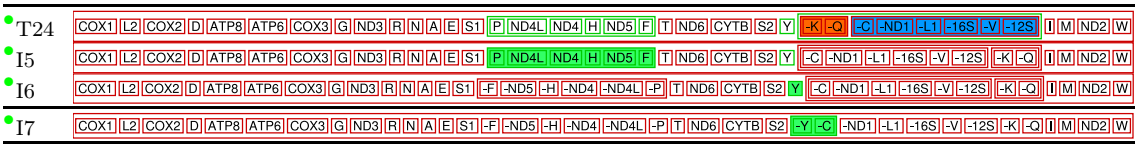


Figure B.17 – Rearrangement scenarios from the connected component shown in Figure 4.21a; from top to bottom: NC_006293-NC_008742, NC_008742-NC_008974

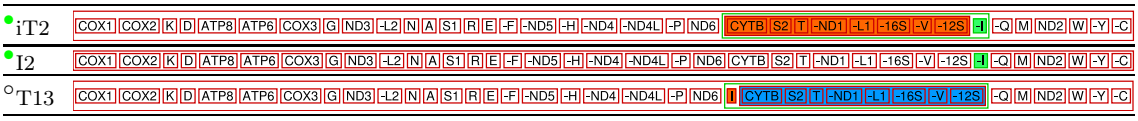


Figure B.18 – Rearrangements from the connected component shown in Figure 4.21b; from top to bottom: NC_005925-NC_005942, NC_005925-NC_010777, and NC_005942-NC_010777

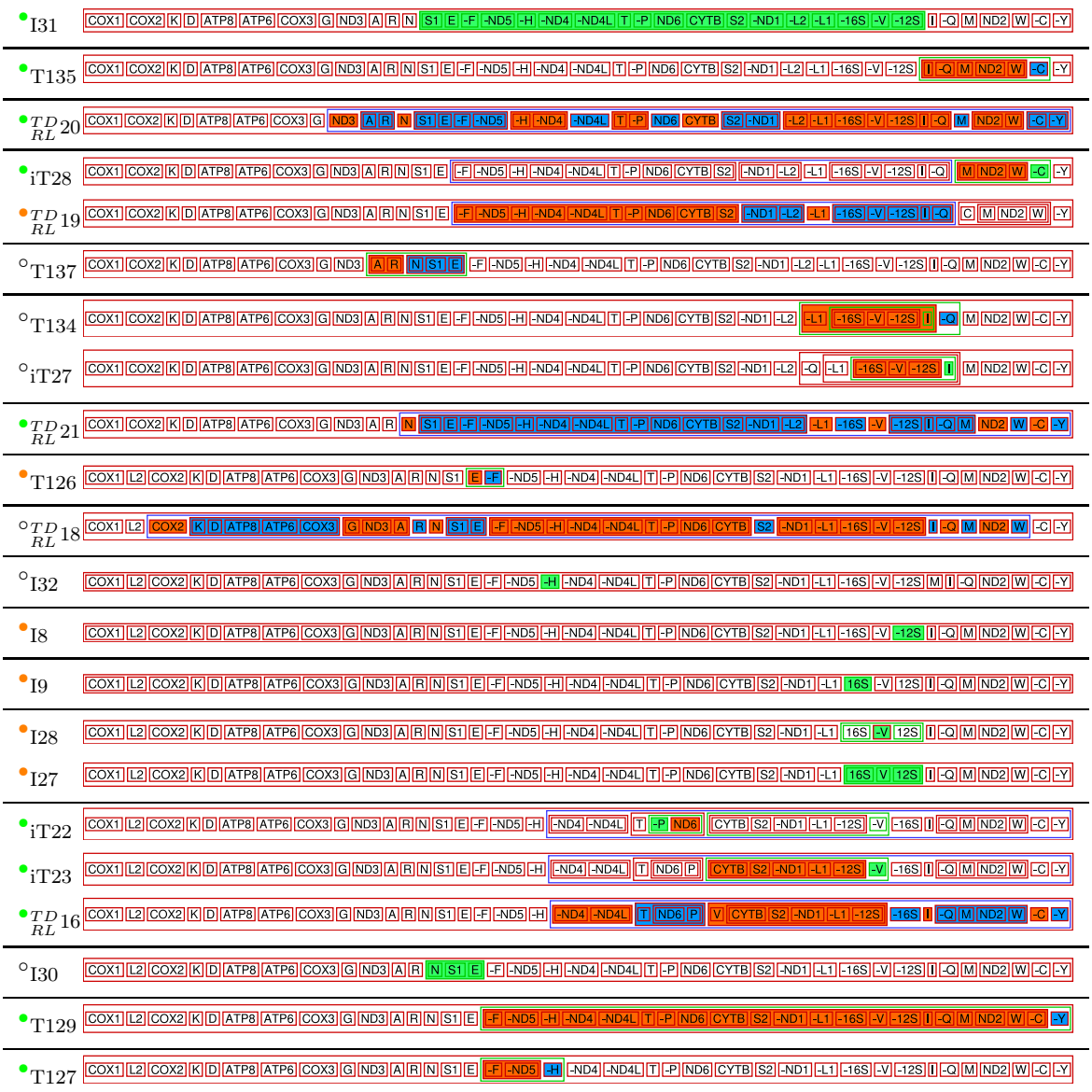


Figure B.19 – Rearrangement scenarios from the connected component shown in Figure 4.22; from top to bottom: NC_002010-NC_008557, NC_002010-NC_002629, NC_002010-NC_005870, NC_002010-NC_002074, NC_002010-NC_010430, NC_002010-NC_010766, NC_002010-NC_009985, NC_000844-NC_003081, NC_000844-NC_004816, NC_002355-NC_008141, NC_000844-NC_012463, NC_006081-NC_012463, NC_006081-NC_007010, NC_007010-NC_011243, NC_000844-NC_011823, NC_000844-NC_007688, and NC_000844-NC_005037

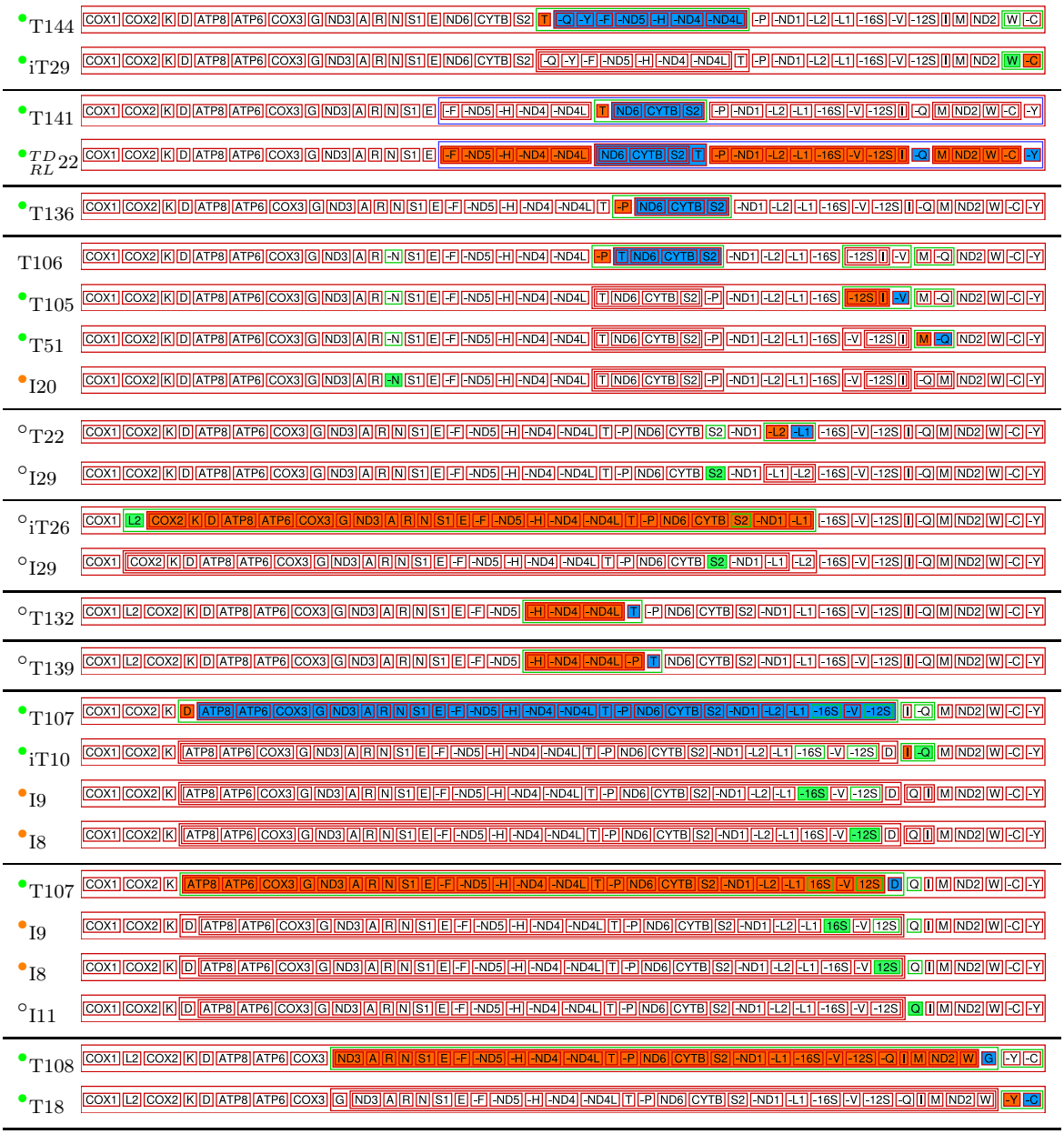


Figure B.20 – Rearrangement scenarios from the connected component shown in Figure 4.23 (left half); from top to bottom: NC_003343-NC_010221, NC_009984-NC_003343, NC_002010-NC_009984, NC_008453-NC_009984, NC_002010-NC_010779, NC_000844-NC_010779, NC_000844-NC_012738, NC_012421-NC_012738, NC_002010-NC_006515, NC_006515-NC_009724, and NC_006280-NC_012459

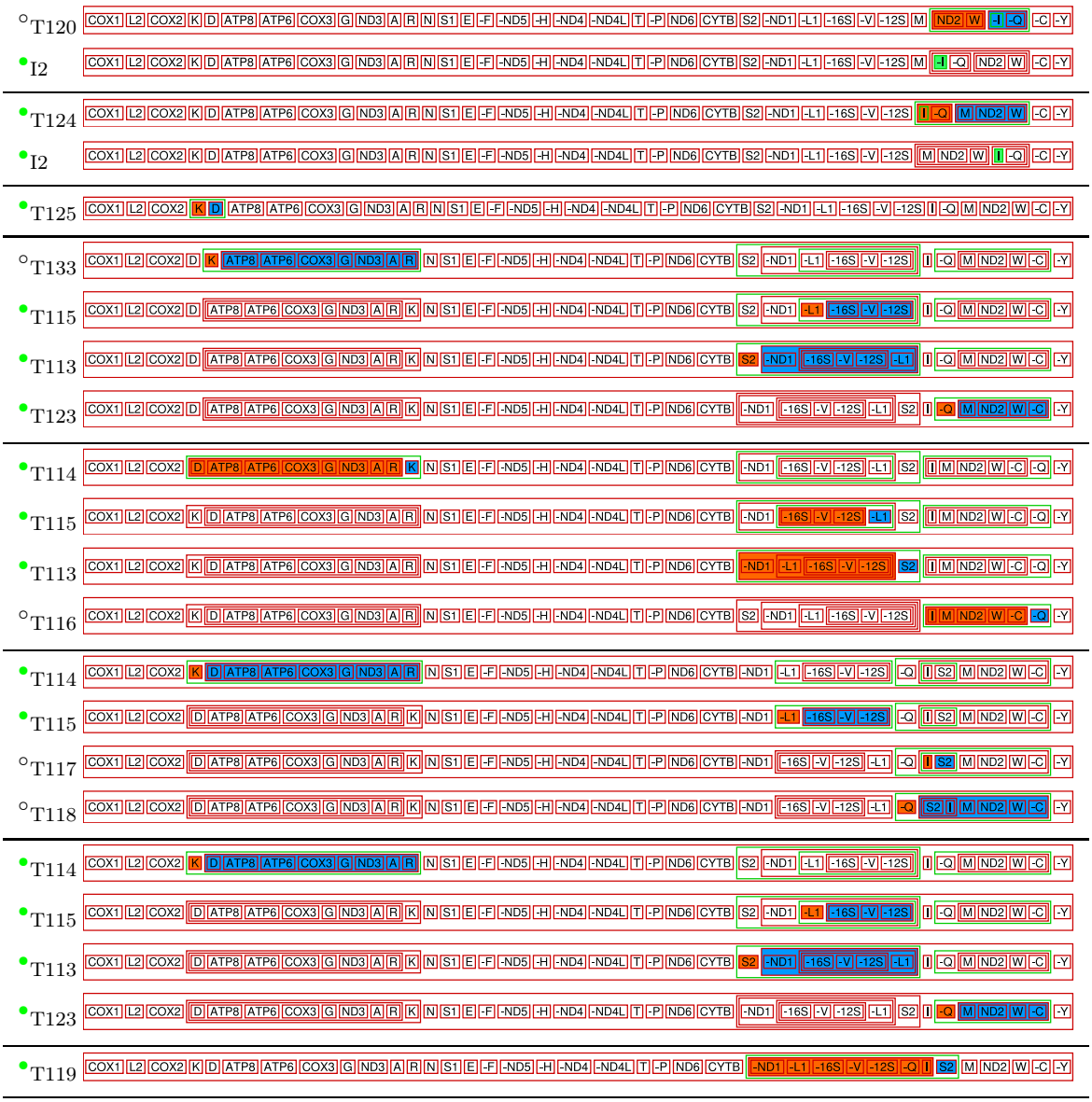


Figure B.21 – Rearrangement scenarios from the connected component shown in Figure 4.23 (bottom right quadrant); from top to bottom: NC_001620-NC_002355, NC_000844-NC_001620, NC_000844-NC_001712, NC_001712-NC_005934, NC_005934-NC_012459, NC_002735-NC_005934, NC_000844-NC_005934, and NC_002735-NC_012459

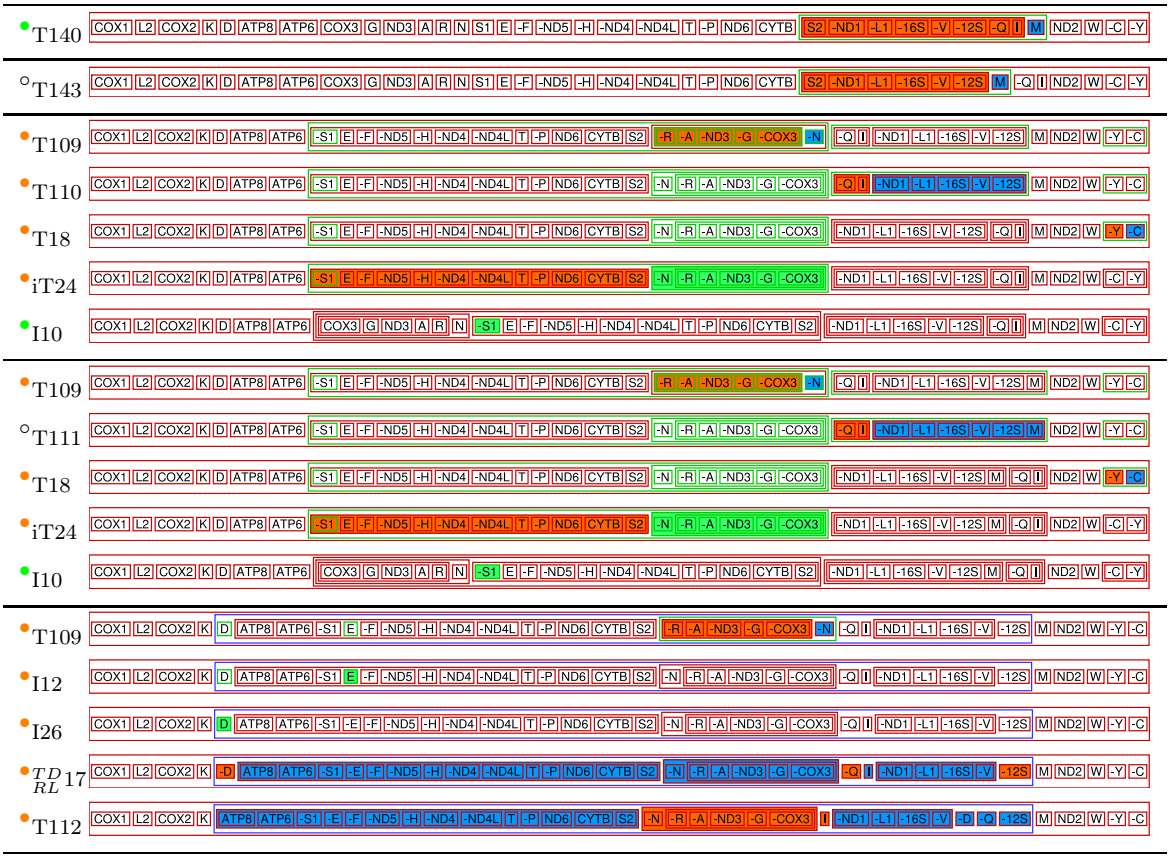


Figure B.22 – Rearrangement scenarios from the connected component shown in Figure 4.23 (at the bottom); from top to bottom: NC_012459-NC_012689, NC_012688-NC_012689, NC_006160-NC_012459, NC_006160-NC_012688, and NC_006160-NC_006279

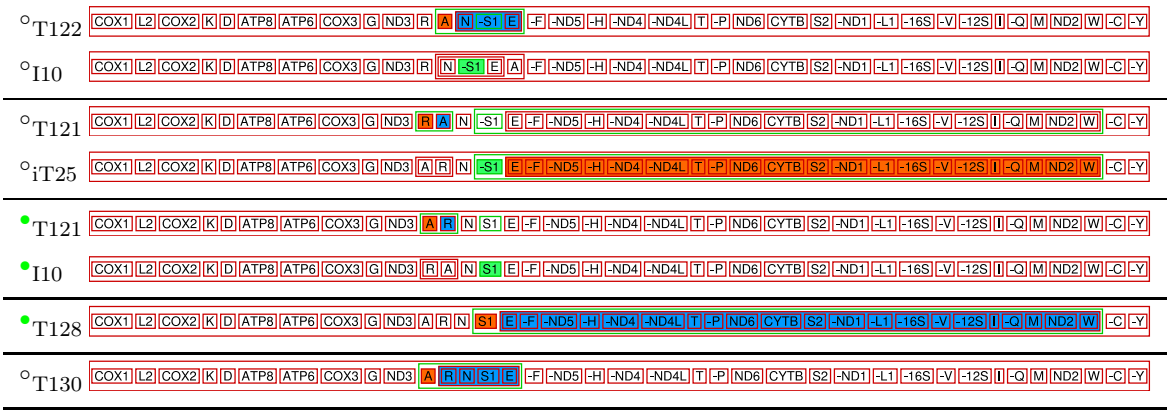


Figure B.23 – Rearrangement scenarios from the connected component shown in Figure 4.23 (top right quadrant); from top to bottom: NC_000875-NC_010532, NC_000875-NC_006158, NC_000844-NC_000875, NC_000844-NC_006158, and NC_000844-NC_010532

Figure B.24 – Rearrangement scenarios including iT21 from the connected component shown in Figure 4.23; from top to bottom: NC_008453-NC_012421, NC_000844-NC_002010, and NC_009724-NC_012459

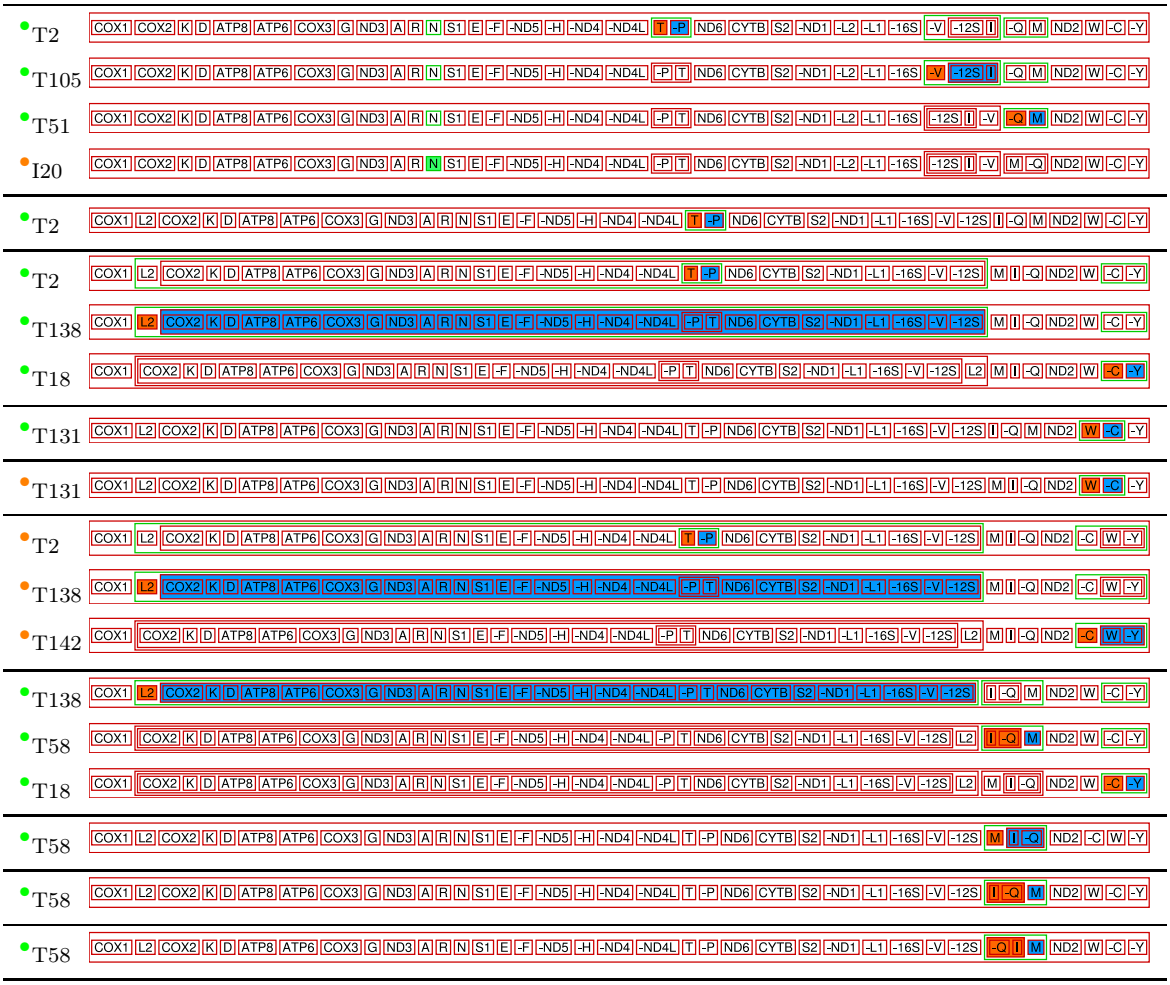


Figure B.25 – Rearrangement scenarios including T2 and T58 from the connected component shown in Figure 4.23; from top to bottom: NC_002010-NC_008453, NC_000844-NC_012421, NC_002355-NC_012708, NC_000844-NC_011277, NC_002355-NC_011128, NC_011128-NC_012708, NC_012421-NC_012708, NC_011128-NC_011277, NC_000844-NC_002355, and NC_012459-NC_012688

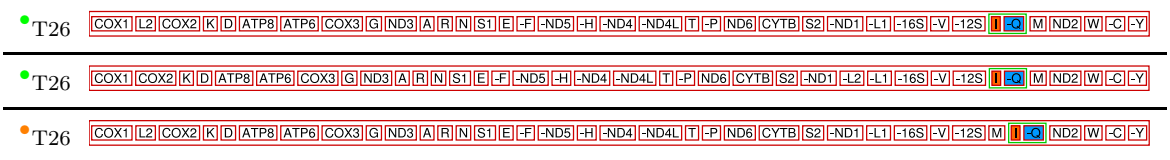


Figure B.26 – Transposition 26 from the connected component shown in Figure 4.23; from top to bottom: NC_000844-NC_012459, NC_002010-NC_009724, and NC_002355-NC_012688

Bibliography

Adam, Z. and Sankoff, D., 2008. The ABCs of MGR with DCJ. *Evolutionary Bioinformatics*, 4:69–74.

Adam, Z., Turmel, M., Lemieux, C., and Sankoff, D., 2007. Common Intervals and Symmetric Difference in a Model-Free Phylogenomics, with an Application to Streptophyte Evolution. *Journal of Computational Biology*, 14(4):436–445.

Ailon, N., Charikar, M., and Newman, A., 2008. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5):1–27.

Ajana, Y., Lefebvre, J.F., Tillier, E.R., and El-Mabrouk, N., 2002. Exploring the Set of All Minimal Sequences of Reversals - An Application to Test the Replication-Directed Reversal Hypothesis. In *Algorithms in Bioinformatics, Second International Workshop, WABI 2002, Proceedings*, volume 2452 of *Lecture Notes in Computer Science*, pages 300–315. Springer.

Akasaki, T., Nikaido, M., Tsuchiya, K., Segawa, S., Hasegawa, M., and Okada, N., 2006. Extensive mitochondrial gene arrangements in coleoid Cephalopoda and their phylogenetic implications. *Molecular Phylogenetics and Evolution*, 38(3):648–658.

Aldous, D. and Diaconis, P., 1986. Shuffling Cards and Stopping Times. *The American Mathematical Monthly*, 93(5):333–348.

Alekseyev, M.A. and Pevzner, P.A., 2007. Are There Rearrangement Hotspots in the Human Genome? *PLoS Computational Biology*, 3(11):e209.

Arndt, A. and Smith, M.J., 1998. Mitochondrial gene rearrangement in the sea cucumber genus Cucumaria. *Molecular Biology and Evolution*, 15(8):9–16.

Arndt, W. and Tang, J., 2007. Improving Inversion Median Computation Using Commuting Reversals and Cycle Information. In *Comparative Genomics, RECOMB 2007 International Workshop, RECOMB-CG 2007, Proceedings*, volume 4751 of *Lecture Notes in Computer Science*, pages 30–44. Springer.

Bachrach, A., Chen, K., Harrelson, C., Mihaescu, R., Rao, S., and Shah, A., 2005. Lower Bounds for Maximum Parsimony with Gene Order Data. In *Comparative Genomics, RECOMB 2005 International Workshop, RCG 2005, Proceedings*, volume 3678 of *Lecture Notes in Computer Science*, pages 1–10. Springer.

Bader, D.A., Moret, B.M.E., and Yan, M., 2001. A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study. In *Algorithms and Data Structures, 7th International Workshop, WADS 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 365–376. Springer.

Bader, M., 2009. Sorting by reversals, block interchanges, tandem duplications, and deletions. *BMC Bioinformatics*, 10(Suppl 1):S9.

Bader, M., Abouelhoda, M., and Ohlebusch, E., 2008. A fast algorithm for the multiple genome rearrangement problem with weighted reversals and transpositions. *BMC Bioinformatics*, 9:516.

Bader, M. and Ohlebusch, E., 2007. Sorting by Weighted Reversals, Transpositions, and Inverted Transpositions. *Journal of Computational Biology*, 14(5):615–636.

Bafna, V. and Pevzner, P.A., 1995. Sorting permutations by tanspositions. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, SODA 1995*, pages 614–623. Society for Industrial and Applied Mathematics.

Bafna, V. and Pevzner, P.A., 1996. Genome Rearrangements and Sorting by Reversals. *SIAM Journal on Computing*, 25(2):272–289.

Bandyopadhyay, P.K., Stevenson, B.J., Cady, M.T., Olivera, B.M., and Wolstenholme, D.R., 2006. Complete mitochondrial DNA sequence of a Conoidean gastropod, Lophiotoma (Xenuroturris) cerithiformis: Gene order and gastropod phylogeny. *Toxicon*, 48(1):29–43.

Bandyopadhyay, P.K., Stevenson, B.J., Ownby, J.P., Cady, M.T., Watkins, M., and Olivera, B.M., 2008. The Mitochondrial Genome of Conus textile, coxI-coxII Intergenic Sequences and Conoidean Evolution. *Molecular Phylogenetics and Evolution*, 46(1):215–223.

Bayer, D. and Diaconis, P., 1992. Trailing the Dovetail Shuffle to its Lair. *The Annals of Applied Probability*, 2(2):294–313.

Béal, M.P., Bergeron, A., Corteel, S., and Raffinot, M., 2004. An Algorithmic View of Gene Teams. *Theoretical Computer Science*, 320(2-3):395–418.

Beard, C.B., Hamm, D.M., and Collins, F.H., 1993. The mitochondrial genome of the mosquito Anopheles gambiae: DNA sequence, genome organization, and comparisons with mitochondrial sequences of other insects. *Insect Molecular Biology*, 2(2):103–124.

Beckenbach, A.T. and Stewart, J.B., 2009. Insect mitochondrial genomics 3: the complete mitochondrial genome sequences of representatives from two neuropteroid orders: a dobsonfly (order Megaloptera) and a giant lacewing and an owlfly (order Neuroptera). *Genome*, 52(1):31–38.

Bender, M.A., Ge, D., He, S., Hu, H., Pinter, R.Y., Skiena, S., and Swidan, F., 2008. Improved Bounds on Sorting by Length-Weighted Reversals. *Journal of Computer and System Sciences*, 74(5):744–774.

Bérard, S., Bergeron, A., and Chauve, C., 2004. Conservation of Combinatorial Structures in Evolution Scenarios. In *Comparative Genomics, RECOMB 2004 International Workshop, RCG 2004, Revised Selected Papers*, volume 3388 of *Lecture Notes in Computer Science*, pages 16–19. Springer.

Bérard, S., Bergeron, A., Chauve, C., and Paul, C., 2007. Perfect sorting by reversals is not always difficult. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):4–16.

Bérard, S., Chateau, A., Chauve, C., Paul, C., and Tannier, E., 2008a. Perfect DCJ Rearrangement. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Computer Science*, pages 158–169. Springer.

Bérard, S., Chauve, C., and Paul, C., 2008b. A more efficient algorithm for perfect sorting by reversals. *Information Processing Letters*, 106(3):90–95.

Bérard, S., Chateau, A., Chauve, C., Paul, C., and Tannier, E., 2009. Computation of perfect DCJ rearrangement scenarios with linear and circular chromosomes. *Journal of Computational Biology*, 16(10):1287–1309.

Bergeron, A., Blanchette, M., Chateau, A., and Chauve, C., 2004a. Reconstructing ancestral gene orders using conserved intervals. In *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Proceedings*, volume 3240 of *Lecture Notes in Computer Science*, pages 14–25. Springer.

Bergeron, A., Chauve, C., de Montgolfier, F., and Raffinot, M., 2005. Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. In *Algorithms - ESA 2005, 13th Annual European Symposium, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 779–790. Springer.

Bergeron, A., Chauve, C., de Montgolfier, F., and Raffinot, M., 2008a. Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. *SIAM Journal on Discrete Mathematics*, 22(3):1022–1039.

Bergeron, A., Corteel, S., and Raffinot, M., 2002a. The Algorithmic of Gene Teams. In *Algorithms in Bioinformatics, Second International Workshop, WABI 2002, Proceedings*, volume 2452 of *Lecture Notes in Computer Science*, pages 464–476. Springer.

Bergeron, A., Heber, S., and Stoye, J., 2002b. Common intervals and sorting by reversals: a marriage of necessity. *Bioinformatics*, 18(Suppl 2):S54–S63.

Bergeron, A., Mixtacki, J., and Stoye, J., 2004b. Reversal Distance without Hurdles and Fortresses. In *Combinatorial Pattern Matching, 15th Annual Symposium, CPM 2004, Proceedings*, volume 3109 of *Lecture Notes in Computer Science*, pages 388–399. Springer.

Bergeron, A., Mixtacki, J., and Stoye, J., 2006. A Unifying View of Genome Rearrangements. In *Algorithms in Bioinformatics, 6th International Workshop, WABI 2006, Proceedings*, volume 4175 of *Lecture Notes in Bioinformatics*, pages 163–173. Springer.

Bergeron, A., Mixtacki, J., and Stoye, J., 2008b. HP Distance Via Double Cut and Join Distance. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Proceedings*, volume 5029 of *Lecture Notes in Computer Science*, pages 56–68. Springer.

Bergeron, A. and Stoye, J., 2006. On the Similarity of Sets of Permutations and Its Applications to Genome Comparison. *Journal of Computational Biology*, 13(7):1340–1354.

Bergeron, A., Mixtacki, J., and Stoye, J., 2008c. On Computing the Breakpoint Reuse Rate in Rearrangement Scenarios. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Computer Science*, pages 226–240. Springer.

Berman, P. and Hannenhalli, S., 1996. Fast sorting by reversal. In *Combinatorial Pattern Matching, 7th Annual Symposium, CPM 96, Proceedings*, volume 1075 of *Lecture Notes in Computer Science*, pages 168–185. Springer.

Bernt, M., Chen, K.Y., Chen, M.C., Chu, A.C., Merkle, D., Wang, H.L., Chao, K.M., and Middendorf, M., 2009a. Finding All Sorting Tandem Duplication Random Loss Operations. *Journal of Discrete Algorithms*. Submitted.

Bernt, M., Chen, M.C., Merkle, D., Wang, H.L., Chao, K.M., and Middendorf, M., 2009b. Finding All Sorting Tandem Duplication Random Loss Operations. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Proceedings*, volume 5577 of *Lecture Notes in Computer Science*, pages 301–313. Springer.

Bernt, M., Merkle, D., and Middendorf, M., 2005. A Parallel Algorithm for Solving the Reversal Median Problem. In *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Revised Selected Papers*, volume 3911 of *Lecture Notes in Computer Science*, pages 1089–1096. Springer.

Bernt, M., Merkle, D., and Middendorf, M., 2006a. Genome Rearrangement Based on Reversals that Preserve Conserved Intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):275–288.

Bernt, M., Merkle, D., and Middendorf, M., 2006b. The Reversal Median Problem, Common Intervals and Mitochondrial Gene Orders. In *Computational Life Sciences II, Second International Symposium, CompLife 2006, Proceedings*, volume 4216 of *Lecture Notes in Bioinformatics*, pages 52–63. Springer.

Bernt, M., Merkle, D., and Middendorf, M., 2007a. A Fast and Exact Algorithm for the Perfect Reversal Median Problem. In *Bioinformatics Research and Applications: Third International Symposium, ISBRA 2007, Proceedings*, volume 4463 of *Lecture Notes in Bioinformatics*, pages 305–316. Springer.

Bernt, M., Merkle, D., and Middendorf, M., 2007b. Using median sets for inferring phylogenetic trees. *Bioinformatics*, 23(2):129–135.

Bernt, M., Merkle, D., and Middendorf, M., 2008a. An Algorithm for Inferring Mitochondrial Genome Rearrangements in a Phylogenetic Tree. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Bioinformatics*, pages 143–157. Springer.

Bernt, M., Merkle, D., and Middendorf, M., 2008b. Solving the Preserving Reversal Median Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):332–347.

Bernt, M., Merkle, D., Ramsch, K., Fritzsch, G., Perseke, M., Bernhard, D., Schlegel, M., Stadler, P.F., and Middendorf, M., 2007c. CREx: inferring genomic rearrangements based on common intervals. *Bioinformatics*, 23(21):2957–2958.

Bhutkar, A., Gelbart, W., and Smith, T., 2007. Inferring genome-scale rearrangement phylogeny and ancestral gene order: a Drosophila case study. *Genome Biology*, 8(11):R236.

Black 4th, W.C. and Roehrdanz, R.L., 1998. Mitochondrial gene order is not conserved in arthropods: prostriate and metastriate tick mitochondrial genomes. *Molecular Biology and Evolution*, 15(12):1772–1785.

Blanchette, M., Bourque, G., and Sankoff, D., 1997. Breakpoint Phylogenies. In *Genome Informatics*, pages 25–34. Universal Academy Press.

Blanchette, M., Kunisawa, T., and Sankoff, D., 1996. Parametric genome rearrangement. *Gene*, 172(1):11–17.

Blanchette, M., Kunisawa, T., and Sankoff, D., 1999. Gene Order Breakpoint Evidence in Animal Mitochondrial Phylogeny. *Journal of Molecular Evolution*, 49(2):193–203.

Bleidorn, C., Podsiadlowski, L., and Bartolomaeus, T., 2006. The complete mitochondrial genome of the orbiniid polychaete Orbinia latreillii (Annelida, Orbiniidae)– A novel gene order for Annelida and implications for annelid phylogeny. *Gene*, 370:96–103.

Böcker, S., Jahn, K., Mixtacki, J., and Stoye, J., 2008. Computation of Median Gene Clusters. In *Research in Computational Molecular Biology, 12th Annual International Conference, RECOMB 2008. Proceedings*, volume 4955 of *Lecture Notes in Computer Science*, pages 331–345. Springer.

Boore, J.L., 1999. Animal mitochondrial genomes. *Nucleic Acids Research*, 27(8):1767–1780.

Boore, J.L., 2000. The duplication/random loss model for gene rearrangement exemplified by mitochondrial genomes of deuterostome animals. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, volume 1 of *Computational Biology Series*, pages 133–147. Kluwer Academic Publishers.

Boore, J.L., 2001. Mitochondrial Genome Rearrangement Guide. Version 6.1.

Boore, J.L., 2006a. The complete sequence of the mitochondrial genome of Nautilus macromphalus (Mollusca: Cephalopoda). *BMC Genomics*, 7:182.

Boore, J.L., 2006b. Mitochondrial data base. Not available since December 2006.

Boore, J.L., Collins, T.M., D, S., L., D.L., and M., B.W., 1995. Deducing the pattern of arthropod phylogeny from mitochondrial DNA rearrangements. *Nature*, 376(6536):163–165.

Boore, J.L., Lavrov, D.V., and Brown, W.M., 1998. Gene translocation links insects and crustaceans. *Nature*, 392(6677):667–668.

Boore, J.L., R., M.J., and Medina, M., 2005. Sequencing and Comparing Whole Mitochondrial Genomes of Animals. *Methods in Enzymology: Producing the Biochemical Data II*, 395:311–348.

Booth, K. and Lueker, G., 1976. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379.

Bourque, G. and Pevzner, P.A., 2002. Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species. *Genome Research*, 12(1):26–36.

Bouvel, M. and Rossin, D., 2009. A variant of the tandem duplication – random loss model of genome rearrangement. *Theoretical Computer Science*, 410(8-10):847–858.

Bouvel, M., Chauve, C., Mishna, M., and Rossin, D., 2009. Average-Case Analysis of Perfect Sorting by Reversals. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Proceedings*, volume 5577 of *Lecture Notes in Computer Science*, pages 314–325. Springer.

Braga, M., Sagot, M.F., Scornavacca, C., and Tannier, E., 2008. Exploring the solution space of sorting by reversals, with experiments and an application to evolution. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):348–356.

Braga, M.D.V. and Stoye, J., 2009. Counting All DCJ Sorting Scenarios. In *Comparative Genomics*, volume 5817 of *Lecture Notes in Computer Science*, pages 36–47. Springe.

Bryant, D., 1998. The complexity of the breakpoint median problem. Technical report, CRM-2579 Centre de recherches mathématiques, Université de Montréal.

Burton, R.S., Byrne, R.J., and Rawson, P.D., 2007. Three divergent mitochondrial genomes from California populations of the copepod Tigriopus californicus. *Gene*, 403(1-2):53–59.

Cameron, S.L., Beckenbach, A.T., Dowton, M., and Whiting, M.F., 2006a. Evidence from Mitochondrial Genomics on Interordinal Relationships in Insects. *Arthropod Systematics & Phylogeny*, 64(1):27–34.

Cameron, S.L., Lambkin, C.L., Barker, S.C., and Whiting, M.F., 2006b. A mitochondrial genome phylogeny of Diptera: whole genome sequence data accurately resolve relationships over broad timescales with high precision. *Systematic Entomology*, 32(1):40–59.

Caprara, A., 2003. The Reversal Median Problem. *INFORMS Journal on Computing*, 15(1):93–113.

Caprara, A., 1999. Sorting Permutations by Reversals and Eulerian Cycle Decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110.

Cha, S.Y., Yoon, H.J., Lee, E.M., Yoon, M.H., Hwang, J.S., Jin, B.R., Han, Y.S., and Kim, I., 2007. The complete nucleotide sequence and gene organization of the mitochondrial genome of the bumblebee, Bombus ignitus (Hymenoptera: Apidae). *Gene*, 392(1-2):206–220.

Chaudhuri, K., Chen, K., Mihaescu, R., and Rao, S., 2006. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 564–570. ACM.

Chauve, C. and Tannier, E., 2008. A Methodological Framework for the Reconstruction of Contiguous Regions of Ancestral Genomes and Its Application to Mammalian Genomes. *PLoS Computational Biology*, 4(11):e1000234.

Cook, C.E., Yue, Q., and Akam, M., 2005. Mitochondrial genomes suggest that hexapods and crustaceans are mutually paraphyletic. *Proceedings of the Royal Society B*, 272(1569):1295–1304.

Cosner, M., Jansen, R.K., Moret, B.M.E., Raubeson, L.A., Wang, L.S., Warnow, T., and Wyman, S., 2000a. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, volume 1 of *Computational Biology Series*, pages 99–121. Kluwer Academic Publishers.

Cosner, M., Jansen, R.K., Moret, B.M.E., Raubeson, L.A., Wang, L.S., Warnow, T., and Wyman, S., 2000b. A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 104–115. AAAI.

Dalevi, D.A., Eriksen, N., Eriksson, K., and Andersson, S.G., 2002. Measuring Genome Divergence in Bacteria: A Case Study Using Chlamydian Data. *Journal of Molecular Evolution*, 55(1):24–36.

Darling, A.E., Miklós, I., and Ragan, M.A., 2008. Dynamics of Genome Rearrangement in Bacterial Populations. *PLoS Genetics*, 4(7):e1000128.

Dávila, S., Piñero, D., Bustos, P., Cevallos, M.A., and Dávila, G., 2005. The mitochondrial genome sequence of the scorpion Centruroides limpidus (Karsch 1879) (Chelicerata; Arachnida). *Gene*, 360(2):92–102.

Diekmann, Y., Sagot, M.F., and Tannier, E., 2007. Evolution under Reversals: Parsimony and Conservation of Common Intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):301–309.

Dobzhansky, T. and Sturtevant, A.H., 1938. Inversions in the Chromosomes of Drosophila Pseudoobscura. *Genetics*, 23(1):28–64.

Downey, R.G. and Fellows, M., 1999. *Parameterized Complexity*. Monographs in Computer Science. Springer.

Dowton, M., Cameron, S.L., Dowavic, J.I., Austin, A.D., and Whiting, M., 2009. Characterization of 67 Mitochondrial tRNA Gene Rearrangements in the Hymenoptera Suggests That Mitochondrial tRNA Gene Position Is Selectively Neutral. *Molecular Biology and Evolution*, 26(7):1607–1617.

Dowton, M., Castro, L.R., Campbell, S.L., Bargon, S.D., and Austin, A.D., 2003. Frequent Mitochondrial Gene Rearrangements at the Hymenopteran nad3-nad5 Junction. *Journal of Molecular Evolution*, 56(5):517–526.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D., 2001. Rank aggregation methods for the Web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 613–622. ACM Press.

Earnest-DeYoung, J.V., Lerat, E., and Moret, B.M.E., 2004. Reversing Gene Erosion - Reconstructing Ancestral Bacterial Genomes from Gene-Content and Order Data. In *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Proceedings*, volume 3240 of *Lecture Notes in Computer Science*, pages 1–13. Springer.

Eisen, J.A., Heidelberg, J.F., White, O., and Salzberg, S.L., 2000. Evidence for symmetric chromosomal inversions around the replication origin in bacteria. *Genome Biology*, 1(6):research0011.1–0011.9.

El-Mabrouk, N., 2000a. Duplication, rearrangement and reconciliation. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, volume 1 of *Computational Biology Series*, pages 537–550. Kluwer Academic Publishers.

El-Mabrouk, N., 2000b. Genome Rearrangement by Reversals and Insertions/Deletions of Contiguous Segments. In *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 222–234. Springer.

Elias, I. and Hartman, T., 2006. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379.

Eres, R., Landau, G.M., and Parida, L., 2003. A Combinatorial Approach to Automatic Discovery of Cluster-Patterns. In *Algorithms in Bioinformatics, Third International Workshop, WABI 2003, Proceedings*, volume 2812 of *Lecture Notes in Computer Science*, pages 139–150. Springer.

Eriksen, N., 2003. *Combinatorial methods in comparative genomics*. Ph.D. thesis, Department of mathematics, Royal Institute of Technology, KTH, Sweden.

Eriksen, N., 2002. $(1 + \epsilon)$-Approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517–529.

Fahrein, K., Talarico, G., Braband, A., and Podsiadlowski, L., 2007. The complete mitochondrial genome of Pseudocellus pearsei (Chelicerata: Ricinulei) and a comparison of mitochondrial gene rearrangements in Arachnida. *BMC Genomics*, 8:386.

Feijão, P.C., Neiva, L.S., Lima de Azeredo-Espin, A.M., and Lessinger, A.C., 2006. AMiGA: the arthropodan mitochondrial genomes accessible database. *Bioinformatics*, 22(7):902–903.

Feng, J.X. and Zhu, D.M., 2007. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms*, 3(3):Article No. 25.

Fenn, J.D., Song, H., Cameron, S.L., and Whiting, M.F., 2008. A preliminary mitochondrial genome phylogeny of Orthoptera (Insecta) and approaches to maximizing phylogenetic signal found within mitochondrial genome data. *Molecular Phylogenetics and Evolution*, 49(1):59–68.

Figeac, M. and Varré, J.S., 2004. Sorting by Reversals with Common Intervals. In *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Proceedings*, volume 3240 of *Lecture Notes in Computer Science*, pages 26–37. Springer.

Fitch, W., 1971. Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20(4):406–416.

Flook, P.K., Rowell, C.H.F., and Gellissen, G., 1995. The sequence, organization, and evolution of the Locusta migratoria mitochondrial genome. *Journal of Molecular Evolution*, 41(6):928–941.

Fonseca, M.M. and Harris, D.J., 2008. Relationship between mitochondrial gene rearrangements and stability of the origin of light strand replication. *Genetics and Molecular Biology*, 30(2):566–574.

Friedrich, M. and Muqim, N., 2003. Sequence and phylogenetic analysis of the complete mitochondrial genome of the flour beetle Tribolium castanaeum. *Molecular Phylogenetics and Evolution*, 26(3):502–512.

Gallut, C. and Barriel, V., 2002. Cladistic coding of genomic maps. *Cladistics*, 18(5):526–536.

Gallut, C., Barriel, V., and Vignes, R., 2000. Gene Order and Phylogenetic Information. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, volume 1 of *Computational Biology Series*, pages 123–132. Kluwer Academic Publishers.

Grande, C., Templado, J., and Zardoya, R., 2008. Evolution of gastropod mitochondrial genome arrangements. *BMC Evolutionary Biology*, 8:61.

Grinstead, C.M. and Snell, J.L., 2006. *Introduction to Probability*. American Mathematical Society. `http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html`.

Hannenhalli, S., Chappey, C., Koonin, E., and Pevzner, P.A., 1995. Genome Sequence Comparison and Scenarios for Gene Rearrangements: A Test Case. *Genomics*, 30(2):299–311.

Hannenhalli, S. and Pevzner, P., 1995a. Transforming Men into Mice (Polynomial Algorithm for Genomic Distance Problem). In *Annual IEEE Symposium on Foundations of Computer Science*, pages 581–592. IEEE Computer Society.

Hannenhalli, S. and Pevzner, P.A., 1995b. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 178–189. ACM.

Hartman, T., 2003. A Simpler 1.5-Approximation Algorithm for Sorting by Transpositions. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Proceedings*, volume 2676 of *Lecture Notes in Computer Science*, pages 156–169. Springer.

Hartman, T. and Shamir, R., 2006. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204(2):275–290.

Hatzoglou, E., Rodakis, G.C., and Lecanidou, R., 1995. Complete sequence and gene organization of the mitochondrial genome of the land snail Albinaria coerulea. *Genetics*, 140(4):1353–1366.

Heber, S., Mayr, R., and Stoye, J., 2009. Common Intervals of Multiple Permutations. *Algorithmica*. To appear.

Heber, S. and Savage, C.D., 2005. Common intervals of trees. *Information Processing Letters*, 93(2):69–74.

Heber, S. and Stoye, J., 2001a. Algorithms for Finding Gene Clusters. In *Algorithms in Bioinformatics, First International Workshop, WABI 2001, Proceedings*, volume 2149 of *Lecture Notes in Computer Science*, pages 252–263. Springer.

Heber, S. and Stoye, J., 2001b. Finding all Common Intervals of k Permutations. In *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001, Proceedings*, volume 2089 of *Lecture Notes in Computer Science*, pages 207–218. Springer.

Hoberman, R. and Durand, D., 2005. The Incompatible Desiderata of Gene Cluster Properties. In *Comparative Genomics, RECOMB 2005 International Workshop, RCG 2005, Proceedings*, volume 3678 of *Lecture Notes in Computer Science*, pages 73–87. Springer.

Hsu, W.L. and McConnell, R.M., 2003. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116.

Hua, J., Li, M., Dong, P., Cui, Y., Xie, Q., and Bu, W., 2008. Comparative and phylogenomic studies on the mitochondrial genomes of Pentatomomorpha (Insecta: Hemiptera: Heteroptera). *BMC Genomics*, 9:610.

Huson, D., Nettles, S., and Warnow, T., 1999. Disk-Covering, a fast converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6(3):369–386.

Hwang, U.W., Friedrich, M., Tautz, D., Park, C.J., and Kim, W., 2001. Mitochondrial protein phylogeny joins myriapods with chelicerates. *Nature*, 413(6852):154–157.

Inoue, J.G., Miya, M., Tsukamoto, K., and Nishida, M., 2001. Complete Mitochondrial DNA Sequence of Conger myriaster (Teleostei: Anguilliformes): Novel Gene Order for Vertebrate Mitochondrial Genomes and the Phylogenetic Implications for Anguilliform Families. *Journal of Molecular Evolution*, 52(4):311–320.

Inoue, J.G., Miya, M., Tsukamoto, K., and Nishida, M., 2003. Evolution of the Deep-Sea Gulper Eel Mitochondrial Genomes: Large-Scale Gene Rearrangements Originated Within the Eels. *Molecular Biology and Evolution*, 20(11):1917–1924.

Interian, Y., 2006. *Models and Algorithms: Application to Satisfiability and Genome Rearrangement Problems*. Ph.D. thesis, Faculty of the Graduate School of Cornell University.

Jameson, D., Gibson, A., Hudelot, C., and Higgs, P., 2003. OGRe: a relational database for comparative analysis of mitochondrial genomes. *Nucleic Acids Research*, 31(1):202–206.

Jennings, R.M. and Halanych, K.M., 2005. Mitochondrial Genomes of Clymenella torquata (Maldanidae) and Riftia pachyptila (Siboglinidae): Evidence for Conserved Gene Order in Annelida. *Molecular Biology and Evolution*, 22(2):210–222.

Kaplan, H., Shamir, R., and Tarjan, R.E., 1999. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892.

Kaplan, H. and Verbin, E., 2003. Efficient Data Structures and a New Randomized Approach for Sorting Signed Permutations by Reversals. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Proceedings*, volume 2676 of *Lecture Notes in Computer Science*, pages 170–185. Springer.

Kececioglu, J. and Sankoff, D., 1995. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1-2):180–210.

Kececioglu, J. and Sankoff, D., 1994. Efficient bounds for oriented chromosome inversion distance. In *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Proceedings*, volume 807 of *Lecture Notes in Computer Science*, pages 307–325. Springer.

Kurabayashi, A. and Ueshima, R., 2000. Complete Sequence of the Mitochondrial DNA of the Primitive Opisthobranch Gastropod Pupa strigosa: Systematic Implication of the Genome Organization. *Molecular Biology and Evolution*, 17(2):266–277.

Labarre, A., 2006. New Bounds and Tractable Instances for the Transposition Distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):380–394.

Landau, G.M., Parida, L., and Weimann, O., 2005. Gene Proximity Analysis Across Whole Genomes via PQ Trees. *Journal of Computational Biology*, 12(10):1289–1306.

Larget, B., Kadane, J.B., and Simon, D.L., 2005. A Bayesian approach to the estimation of ancestral genome arrangements. *Molecular Phylogenetics and Evolution*, 36(2):214–223.

Laslett, D. and Canbäck, B., 2008. ARWEN: a program to detect tRNA genes in metazoan mitochondrial nucleotide sequences. *Bioinformatics*, 24(2):172–175.

Lathe, W.C., Snel, B., and Bork, P., 2000. Gene context conservation of a higher order than operons. *Trends in Biochemical Sciences*, 25(10):474–479.

Lavrov, D.V., Boore, J.L., and Brown, W.M., 2002. Complete mtDNA Sequences of Two Millipedes Suggest a New Model for Mitochondrial Gene Rearrangements: Duplication and Nonrandom Loss. *Molecular Biology and Evolution*, 19(2):163–169.

Lavrov, D.V., Brown, W.M., and Boore, J.L., 2004. Phylogenetic position of the Pentastomida and (pan)crustacean relationships. *Proceedings of the Royal Society B*, 271(1538):537–544.

Lee, E.S., Shin, K.S., Kim, M.S., Park, H., Cho, S., and Kim, C.B., 2006. The mitochondrial genome of the smaller tea tortrix Adoxophyes honmai (Lepidoptera: Tortricidae). *Gene*, 373:52–57.

Lefebvre, J.F., El-Mabrouk, N., Tillier, E., and Sankoff, D., 2003. Detection and validation of single gene inversions. *Bioinformatics*, 19(Suppl. 1):i190–i196.

Lenne, R., Solnon, C., Stützle, T., Tannier, E., and Birattari, M., 2008. Reactive Stochastic Local Search Algorithms for the Genomic Median Problem. In *Evolutionary Computation in Combinatorial Optimization, 8th European Conference, EvoCOP 2008, Proceedings*, volume 4972 of *Lecture Notes in Computer Science*, pages 266–276. Springer.

Lim, J.T. and Hwang, U.W., 2006. The Complete Mitochondrial Genome of Pollicipes mitella (Crustacea, Maxillopoda, Cirripedia): Non-Monophylies of Maxillopoda and Crustacea. *Molecules and Cells*, 22(3):314–322.

Littlewood, D.T.J., Smith, A.B., Clough, K.A., and Emson, R.H., 1997. The interrelationships of the echinoderm classes: morphological and molecular evidence. *Biological Journal of the Linnean Society*, 61(3):409–438.

Lowe, T.M. and Eddy, S.R., 1997. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Research*, 25(5):955–964.

Luc, N., Risler, J.L., Bergeron, A., and M., R., 2003. Gene Teams: A New Formalization of Gene Clusters For Comparative Genomics. *Computational Biology and Chemistry*, 27(1):59–67.

Ma, C., Liu, C., Yang, P., and Kang, L., 2009. The complete mitochondrial genomes of two band-winged grasshoppers, Gastrimargus marmoratus and Oedaleus asiaticus. *BMC Genomics*, 10:156.

Ma, J., Zhang, L., Bernard, B., Raney, B., Burhans, R., Kent, W., Blanchette, M., Haussler, D., and Miller, W., 2006. Reconstructing contiguous regions of an ancestral genome. *Genome Research*, 16(12):1557–1565.

Mabuchi, K., Miya, M., Satoh, T.P., Westneat, M.W., and Nishida, M., 2004. Gene Rearrangements and Evolution of tRNA Pseudogenes in the Mitochondrial Genome of the Parrotfish (Teleostei: Perciformes: Scaridae). *Journal of Molecular Evolution*, 59(3):287–297.

Marcotte, E.M., Pellegrini, M., Ng, H.L., Rice, D.W., Yeates, T.O., and Eisenberg, D., 1999. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–753.

Marron, M., Swenson, K.M., and Moret, B.M.E., 2003. Genomic Distances under Deletions and Insertions. In *Computing and Combinatorics, 9th Annual International Conference, COCOON 2003, Proceedings*, volume 2697 of *Lecture Notes in Computer Science*, pages 537–547. Springer.

Masta, S.E. and Boore, J.L., 2004. The Complete Mitochondrial Genome Sequence of the Spider Habronattus oregonensis Reveals Rearranged and Extremely Truncated tRNAs. *Molecular Biology and Evolution*, 21(5):893–902.

Mauro, D.S., Gower, D.J., Zardoya, R., and Wilkinson, M., 2006. A Hotspot of Gene Order Rearrangement by Tandem Duplication and Random Loss in the Vertebrate Mitochondrial Genome. *Molecular Biology and Evolution*, 23(1):227–234.

Maynard, B.T., Kerr, L.J., McKiernan, J.M., Jansen, E.S., and Hanna, P.J., 2005. Mitochondrial DNA Sequence and Gene Organization in Australian Backup Abalone Haliotis Rubra (Leach). *Marine Biotechnology*, 7(6):645–658.

Miklós, I. and Hein, J., 2005. Genome Rearrangement in Mitochondria and Its Computational Biology. In *Comparative Genomics, RECOMB 2005 International Workshop, RCG 2005, Proceedings*, volume 3388 of *Lecture Notes in Computer Science*, pages 85–96. Springer.

Miller, A.D., Nguyen, T.T.T., Burridge, C.P., and Austin, C.M., 2004. Complete mitochondrial DNA sequence of the Australian freshwater crayfish, Cherax destructor (Crustacea: Decapoda: Parastacidae): a novel gene order revealed. *Gene*, 331:65–72.

Miya, M., Kawaguchi, A., and Nishida, M., 2001. Mitogenomic Exploration of Higher Teleostean Phylogenies: A Case Study for Moderate-Scale Evolutionary Genomics with 38 Newly Determined Complete Mitochondrial DNA Sequences. *Molecular Biology and Evolution*, 18(11):1993–2009.

Miya, M. and Nishida, M., 1999. Organization of the Mitochondrial Genome of a Deep-Sea Fish, Gonostoma gracile (Teleostei: Stomiiformes): First Example of Transfer RNA Gene Rearrangements in Bony Fishes. *Marine Biotechnology*, 1(5):416–426.

Miya, M., Satoh, T.P., and Nishida, M., 2005. The phylogenetic position of toadfishes (order Batrachoidiformes) in the higher ray-finned fish as inferred from partitioned Bayesian analysis of 102 whole mitochondrial genome sequences. *Biological Journal of the Linnean Society*, 85(3):289–306.

Miya, M., Takeshima, H., Endo, H., Ishiguro, N.B., Inoue, J.G., Mukai, T., Satoh, T.P., Yamaguchi, M., Kawaguchi, A., Mabuchi, K., Shirai, S.M., and Nishida, M., 2003. Major patterns of higher teleostean phylogenies: a new perspective based on 100 complete mitochondrial DNA sequences. *Molecular Phylogenetics and Evolution*, 26(1):121–138.

Moret, B.M.E., Siepel, A.C., Tang, J., and Liu, T., 2002a. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *Algorithms in Bioinformatics, Second International Workshop, WABI 2002, Proceedings*, volume 2452 of *Lecture Notes in Computer Science*, pages 521–536. Springer.

Moret, B.M.E., Tang, J., Wang, L.S., and Warnow, T., 2002b. Steps toward accurate reconstructions of phylogenies from gene-order data. *Journal of Computer and System Sciences*, 65(3):508–525.

Moret, B., Wyman, S., Bader, D., Warnow, T., and Yan, M., 2001. A new implementation and detailed study of breakpoint analysis. In *Proceedings of the 6th Pacific Symposium on Biocomputing (PSB 2001)*, pages 583–594.

Nadeau, J.H. and Taylor, B.A., 1984. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences of the United States of America*, 81(3):814–818.

Nardi, F., Carapelli, A., Fanciulli, P.P., Dallai, R., and Frati, F., 2001. The Complete Mitochondrial DNA Sequence of the Basal Hexapod Tetrodontophora bielanensis: Evidence for Heteroplasmy and tRNA Translocations. *Molecular Biology and Evolution*, 18(7):1293–1304.

Negrisolo, E., Minelli, A., and Valle, G., 2004. Extensive Gene Order Rearrangement in the Mitochondrial Genome of the Centipede Scutigera coleoptrata. *Journal of Molecular Evolution*, 58(4):413–423.

Ohlebusch, E., Abouelhoda, M., and Hockel, K., 2007. A linear time algorithm for the inversion median problem in circular bacterial genomes. *Journal of Discrete Algorithms*, 5(4):637–646.

Ouangraoua, A. and Bergeron, A., 2009. Parking Functions, Labeled Trees and DCJ Sorting Scenarios. In *Comparative Genomics*, volume 5817 of *Lecture Notes in Computer Science*, pages 24–35. Springer.

Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G.D., and Maltsev, N., 1999. The use of gene clusters to infer functional coupling. *Proceedings of the National Academy of Sciences of the United States of America*, 96(6):2896–2901.

Ozery-Flato, M. and Shamir, R., 2008. Sorting Genomes with Centromeres by Translocations. *Journal of Computational Biology*, 15(7):793–812.

Parida, L., 2006. Using PQ Structures for Genomic Rearrangement Phylogeny. *Journal of Computational Biology*, 13(10):1685–1700.

Park, S.J., Lee, Y.S., and Hwang, U.W., 2007. The complete mitochondrial genome of the sea spider Achelia bituberculata (Pycnogonida, Ammotheidae): arthropod ground pattern of gene arrangement. *BMC Genomics*, 8:343.

Pe'er, I. and Shamir, R., 1998. The median problems for breakpoints are NP-complete. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(71).

Peng, Q., Pevzner, P.A., and Tesler, G., 2006. The Fragile Breakage versus Random Breakage Models of Chromosome Evolution. *PLoS Computational Biology*, 2(2):e14.

Perseke, M., Fritzsch, G., Ramsch, K., Bernt, M., Merkle, D., Middendorf, M., Bernhard, D., Stadler, P.F., and Schlegel, M., 2008. Evolution of Mitochondrial Gene Orders in Echinoderms. *Molecular Phylogenetics and Evolution*, 47(2):855–864.

Pevzner, P. and Tesler, G., 2003. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 100(13):7672–7677.

Pevzner, P.A., 2000. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press.

Podsiadlowski, L., 2006. The mitochondrial genome of the bristletail Petrobius brevistylis (Archaeognatha: Machilidae). *Insect Molecular Biology*, 15(3):253–258.

Podsiadlowski, L., Kohlhagen, H., and Koch, M., 2007. The complete mitochondrial genome of Scutigerella causeyae (Myriapoda: Symphyla) and the phylogenetic position of Symphyla. *Molecular Phylogenetics and Evolution*, 45(1):251–260.

Pruitt, K.D., Tatusova, T., and Maglott, D.R., 2007. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Research*, 35(Database issue):D61–D65.

Qiu, Y., Song, D., Zhou, K., and Sun, H., 2005. The Mitochondrial Sequences of Heptathela hangzhouensis and Ornithoctonus huwena Reveal Unique Gene Arrangements and Atypical tRNAs. *Journal of Molecular Evolution*, 60(1):57–71.

Ramsch, K., 30.1.2007. *Methoden zur Analyse von Genanordnungen unter besonderer Berücksichtigung von Duplikationsmutationen*. Master's thesis, University of Leipzig.

Ren, J., Liu, X., Zhang, G., L., B., and Guo, X., 2009. "Tandem duplication-random loss" is not a real feature of oyster mitochondrial genomes. *BMC Genomics*, 10:84.

Rokas, A. and Holland, P.W.H., 2000. Rare genomic changes as a tool for phylogenetics. *Trends in Ecology & Evolution*, 15(11):454–459.

Sagot, M.F. and Tannier, E., 2005. Perfect Sorting by Reversals. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 42–51. Springer.

Saitou, N. and Nei, M., 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425.

Salvato, P., Simonato, M., Battisti, A., and Negrisolo, E., 2008. The complete mitochondrial genome of the bag-shelter moth Ochrogaster lunifer (Lepidoptera, Notodontidae). *BMC Genomics*, 9:331.

Sankoff, D., 1999. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917.

Sankoff, D., 2002. Short inversions and conserved gene cluster. *Bioinformatics*, 18(10):1305–1308.

Sankoff, D., 2006. The Signal in the Genomes. *PLoS Computational Biology*, 2(4):e35.

Sankoff, D. and Blanchette, M., 1997. The Median Problem for Breakpoints in Comparative Genomics. In *Computing and Combinatorics, Third Annual International Conference, COCOON '97, Proceedings*, volume 1276 of *Lecture Notes in Computer Science*, pages 251–264. Springer.

Sankoff, D. and Blanchette, M., 1998. Multiple Genome Rearrangement and Breakpoint Phylogeny. *Journal of Computational Biology*, 5(3):555–570.

Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B.F., and Cedergren, R., 1992. Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences of the United States of America*, 89(14):6575–6579.

Sankoff, D., Lefebvre, J.F., Tillier, E.R.M., Maler, A., and El-Mabrouk, N., 2004. The Distribution of Inversion Lengths in Bacteria. In *Comparative Genomics, RECOMB 2004 International Workshop, RCG 2004, Revised Selected Papers*, volume 3388 of *Lecture Notes in Computer Science*, pages 97–108. Springer.

Sankoff, D., Sundaram, G., and Kececioglu, J.D., 1996. Steiner Points in the Space of Genome Rearrangements. *International Journal of Foundations of Computer Science*, 7(1):1–9.

Sankoff, D. and Trinh, P., 2005. Chromosomal breakpoint re-use in genome sequence rearrangement. *Journal of Computational Biology*, 12(6):812–821.

Sankoff, D., 1992. Edit distance for genome comparison based on non-local operations. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Proceedings*, volume 644 of *Lecture Notes in Computer Science*, pages 121–135. Springer.

Sankoff, D. and Nadeau, J.H., 2003. Chromosome rearrangements in evolution: From gene order to genome sequence and back. *Proceedings of the National Academy of Sciences of the United States of America*, 100(20):11188–11189.

Schwenk, A.J., 1986. Elementary Problem: E3143. *The American Mathematical Monthly*, 93(4):299.

Schwenk, A.J., 1988. E3143. *The American Mathematical Monthly*, 95(4):352.

Scouras, A., Beckenbach, K., Arndt, A., and Smith, M.J., 2004. Complete mitochondrial genome DNA sequence for two ophiuroids and a holothuroid: the utility of protein gene sequence and gene maps in the analyses of deep deuterostome phylogeny. *Molecular Phylogenetics and Evolution*, 31(1):50–65.

Sémon, M. and Duret, L., 2006. Evolutionary Origin and Maintenance of Coexpressed Gene Clusters in Mammals. *Molecular Biology and Evolution*, 23(9):1715–1723.

Shao, R., Aoki, Y., Mitani, H., Tabuchi, N., Barker, S.C., and Fukunaga, M., 2004. The mitochondrial genomes of soft ticks have an arrangement of genes that has remained unchanged for over 400 million years. *Insect Molecular Biology*, 13(3):219–224.

Shao, R., Campbell, N.J.H., Schmidt, E.R., and Barker, S.C., 2001. Increased Rate of Gene Rearrangement in the Mitochondrial Genomes of Three Orders of Hemipteroid Insects. *Molecular Biology and Evolution*, 18(9):1828–1832.

Shen, X., Sun, M., Wu, Z., Tian, M., Cheng, H., Zhao, F., and Meng, X., 2009. The complete mitochondrial genome of the ridgetail white prawn Exopalaemon carinicauda Holthuis, 1950 (Crustacean: Decapoda: Palaemonidae) revealed a novel rearrangement of tRNA genes. *Gene*, 437(1-2):1–8.

Shih, W.K. and Hsu, W.L., 1999. A new planarity test. *Theoretical Computer Science*, 223(1-2):179–191.

Siepel, A.C., 2001. *Exact Algorithms for the Reversal Median Problem.* Master's thesis, University of New Mexico.

Siepel, A.C., 2002. An algorithm to find all sorting reversals. In *RECOMB '02: Proceedings of the sixth annual international conference on Computational biology*, pages 281–290. ACM.

Siepel, A.C., 2003. An Algorithm to Enumerate Sorting Reversals for Signed Permutations. *Journal of Computational Biology*, 10(3-4):575–597.

Siepel, A.C. and Moret, B.M.E., 2001. Finding an Optimal Inversion Median: Experimental Results. In *Algorithms in Bioinformatics, First International Workshop, WABI 2001, Proceedings*, volume 2149 of *Lecture Notes in Computer Science*, pages 189–203. Springer.

Stoye, J. and Wittler, R., 2009. A Unified Approach for Reconstructing Ancient Gene Clusters. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):387–400.

Sun, H., Zhou, K., and Song, D., 2005. Mitochondrial genome of the Chinese mitten crab Eriocheir japonica sinenesis (Brachyura: Thoracotremata: Grapsoidea) reveals a novel gene order and two target regions of gene rearrangements. *Gene*, 349:207–217.

Suyama, M. and Bork, P., 2001. Evolution of prokaryotic gene order: genome rearrangements in closely related species. *Trends in Genetics*, 17(1):10–13.

Swenson, K.M., Marron, M., Earnest-DeYoung, J.V., and Moret, B.M.E., 2006. Approximating the true evolutionary distance between two genomes. *ACM Journal of Experimental Algorithmics*, 12:Article No. 3.5.

Tang, J. and Moret, B.M.E., 2003a. Scaling up accurate phylogenetic reconstruction from gene-order data. *Bioinformatics*, 19(Suppl 1):305–312.

Tang, J., Moret, B.M.E., Cui, L., and dePamphilis, C.W., 2004. Phylogenetic reconstruction from arbitrary gene-order data. In *4th IEEE International Symposium on BioInformatics and BioEngineering (BIBE 2004)*, pages 592–599. IEEE Computer Society.

Tang, J. and Wang, L.S., 2005. Improving Genome Rearrangement Phylogeny Using Sequence-Style Parsimony. In *Fifth IEEE International Symposium on Bioinformatic and Bioengineering (BIBE 2005)*, pages 137–144. IEEE Computer Society.

Tang, J. and Moret, B.M.E., 2003b. Phylogenetic Reconstruction from Gene-Rearrangement Data with Unequal Gene Content. In *Algorithms and Data Structures, 8th International Workshop, WADS 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 37–46. Springer.

Tannier, E., Bergeron, A., and Sagot, M.F., 2007. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888.

Tannier, E., Zheng, C., and Sankoff, D., 2009. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10:120.

Taylor, M.F., McKechnie, S.W., Pierce, N., and Kreitman, M., 1993. The lepidopteran mitochondrial control region: structure and evolution. *Molecular Biology and Evolution*, 10(6):1259–1272.

Thao, M.L., Baumann, L., and Baumann, P., 2004. Organization of the mitochondrial genomes of whiteflies, aphids, and psyllids (Hemiptera, Sternorrhyncha). *BMC Evolutionary Biology*, 4:25.

Tillier, E.R.M. and Collins, R.A., 2000. Genome rearrangement by replication-directed translocation. *Nature Genetics*, 26(2):195–197.

Uno, T. and Yagiura, M., 2000. Fast Algorithms to Enumerate All Common Intervals of Two Permutations. *Algorithmica*, 26(2):290–309.

Valverde, J.R., Batuecas, B., Moratilla, C., Marco, R., and Garesse, R., 1994. The complete mitochondrial DNA sequence of the crustacean Artemia franciscana. *Journal of Molecular Evolution*, 39(4):400–408.

Wang, L.S., Warnow, T., Moret, B.M.E., Jansen, R.K., and Raubeson, L.A., 2006. Distance-Based Genome Rearrangement Phylogeny. *Journal of Molecular Evolution*, 63(4):473–483.

Watterson, G.A., Ewens, W.J., Hall, T.E., and Morgan, A., 1982. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1–7.

Webster, B.L., Copley, R.R., Jenner, R.A., Mackenzie-Dodds, J.A., Bourlat, S.J., Rota-Stabelli, O., Littlewood, D.T.J., and Telford, M.J., 2006. Mitogenomics and phylogenomics reveal priapulid worms as extant models of the ancestral Ecdysozoan. *Evolution & Development*, 8(6):502–510.

Wei, S.J., M., S., H., H.J., Sharkey, M., and Chen, X.X., 2009. The complete mitochondrial genome of Diadegma semiclausum (Hymenoptera: Ichneumonidae) indicates extensive independent evolutionary events. *Genome*, 52(4):308–319.

Woo, H.J., Lee, Y.S., Park, S.J., Lim, J.T., Jang, K.H., Choi, E.H., Choi, Y.G., and Hwang, U.W., 2007. Complete Mitochondrial Genome of a Troglobite Millipede Antrokoreana gracilipes (Diplopoda, Juliformia, Julida), and Juliformian Phylogeny. *Molecules and Cells*, 23(2):182–191.

Wyman, S.K., Jansen, R.K., and Boore, J.L., 2004. Automatic annotation of organellar genomes with DOGMA. *Bioinformatics*, 20(17):3252–3255.

Xie, G., Keyhani, N.O., Bonner, C.A., and Jensen, R.A., 2003. Ancient Origin of the Tryptophan Operon and the Dynamics of Evolutionary Change. *Microbiology and Molecular Biology Reviews*, 67(3):303–342.

Xu, A.W., 2008. A fast and exact algorithm for the median of three problem - a graph decomposition approach. In *Research in Computational Molecular Biology, 12th Annual International Conference, RECOMB 2008, Proceedings*, volume 5267 of *Lecture Notes in Computer Science*, pages 184–197. Springer.

Xu, A.W. and Sankoff, D., 2008. Decompositions of Multiple Breakpoint Graphs and Rapid Exact Solutions to the Median Problem. In *Algorithms in Bioinformatics, 8th International Workshop, WABI 2008, Proceedings*, volume 5251 of *Lecture Notes in Computer Science*, pages 25–37. Springer.

Yamauchi, M.M., Miya, M.U., J., M.R., and Nishida, M., 2004. PCR-Based Approach for Sequencing Mitochondrial Genomes of Decapod Crustaceans, with a Practical Example from Kuruma Prawn (Marsupenaeus japonicus). *Marine Biotechnology*, 6(5):419–429.

Yamauchi, M.M., Miya, M.U., and Nishida, M., 2003. Complete mitochondrial DNA sequence of the swimming crab, Portunus trituberculatus (Crustacea: Decapoda: Brachyura). *Gene*, 311:129–135.

Yamazaki, N., Ueshima, R., Terrett, J.A., Yokobori, S.I., Kaifu, M., Segawa, R., Kobayashi, T., Numachi, K.I., Ueda, T., Nishikawa, K., Watanabe, K., and

Thomas, R.H., 1997. Evolution of Pulmonate Gastropod Mitochondrial Genomes: Comparisons of Gene Organizations of Euhadra, Cepaea and Albinaria and Implications of Unusual Trna Secondary Structures. *Genetics*, 145(3):749–758.

Yancopoulos, S., Attie, O., and Friedberg, R., 2005. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346.

Yancopoulos, S. and Friedberg, R., 2008. Sorting Genomes with Insertions, Deletions and Duplications by DCJ. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Computer Science*, pages 170–183. Springer.

Zhang, M. and Leong, H.W., 2008. Gene Team Tree: A Compact Representation of All Gene Teams. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Computer Science*, pages 100–112. Springer.

Zhao, H. and Bourque, G., 2007. Recovering True Rearrangement Events on Phylogenetic Trees. In *Comparative Genomics, RECOMB 2007 International Workshop, RECOMB-CG 2007, Proceedings*, volume 4751 of *Lecture Notes in Computer Science*, pages 149–161. Springer.

Zhu, Q., Adam, Z., Choi, V., and Sankoff, D., 2009. Generalized Gene Adjacencies, Graph Bandwidth and Clusters in Yeast Evolution. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):213–220.

Curriculum vitae

Zur Person

Name:	Matthias Bernt
Anschrift:	Meißner Str. 21, 04315 Leipzig
Geboren:	16. Januar 1979 in Gera

Wissenschaftlicher Werdegang

1997	Abitur am Karl-Theodor-Liebe-Gymnasium in Gera
1998 - 2004	Diplomstudium Informatik an der Universität Leipzig Spezialisierung: Angewandte Informatik Diplomarbeit: *Strukturerhaltende Rekonstruktion von Phylogenien auf der Basis von Genanordnungsdaten*; Betreuer: Prof. Dr. M. Middendorf
2004 - 2005	Promotionsstipendium des Graduiertenkolleg Wissensrepräsentation
2006 - 2008	Wissenschaftlicher Mitarbeiter im Deep Metazoan Phyologeny Projekt (DFG SPP 1174)
2008 -today	Wissenschaftlicher Mitarbeiter am Lehrstuhl für *Parallelverarbeitung und Komplexe Systeme* an der Universität Leipzig

Author's publications

1. Bernt, M., Merkle, D., and Middendorf, M., 2005. A Parallel Algorithm for Solving the Reversal Median Problem. In *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Revised Selected Papers*, volume 3911 of *Lecture Notes in Computer Science*, pages 1089–1096. Springer.

2. Bernt, M., Merkle, D., and Middendorf, M., 2006. Genome Rearrangement Based on Reversals that Preserve Conserved Intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):275–288.

3. Bernt, M., Merkle, D., and Middendorf, M., 2006. The Reversal Median Problem, Common Intervals and Mitochondrial Gene Orders. In *Computational Life Sciences II, Second International Symposium, CompLife 2006, Proceedings*, volume 4216 of *Lecture Notes in Bioinformatics*, pages 52–63. Springer.

4. Bernt, M., Merkle, D., and Middendorf, M., 2007. A Fast and Exact Algorithm for the Perfect Reversal Median Problem. In *Bioinformatics Research and Applications: Third International Symposium, ISBRA 2007, Proceedings*, volume 4463 of *Lecture Notes in Bioinformatics*, pages 305–316. Springer.

5. Bernt, M., Merkle, D., and Middendorf, M., 2007. Using median sets for inferring phylogenetic trees. *Bioinformatics*, 23(2):129–135.

6. Bernt, M., Merkle, D., and Middendorf, M., 2008. An Algorithm for Inferring Mitochondrial Genome Rearrangements in a Phylogenetic Tree. In *Comparative Genomics, International Workshop, RECOMB-CG 2008, Proceedings*, volume 5267 of *Lecture Notes in Bioinformatics*, pages 143–157. Springer.

7. Bernt, M., Merkle, D., and Middendorf, M., 2008. Solving the Preserving Reversal Median Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):332–347.

8. Bernt, M., Merkle, D., Ramsch, K., Fritzsch, G., Perseke, M., Bernhard, D., Schlegel, M., Stadler, P.F., and Middendorf, M., 2007. CREx: inferring genomic rearrangements based on common intervals. *Bioinformatics*, 23(21):2957–2958.

9. Perseke, M., Fritzsch, G., Ramsch, K., Bernt, M., Merkle, D., Middendorf, M., Bernhard, D., Stadler, P.F., and Schlegel, M., 2008. Evolution of Mitochondrial Gene Orders in Echinoderms. *Molecular Phylogenetics and Evolution*, 47(2):855–864.

10. Bernt, M., Chen, M.C., Merkle, D., Wang, H.L., Chao, K.M., and Middendorf, M., 2009. Finding All Sorting Tandem Duplication Random Loss Operations. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Proceedings*, volume 5577 of *Lecture Notes in Computer Science*, pages 301–313. Springer.

11. Bernt, M., Chen, K.Y., Chen, M.C., Chu, A.C., Merkle, D., Wang, H.L., Chao, K.M., and Middendorf, M., 2009. Finding All Sorting Tandem Duplication Random Loss Operations. *Journal of Discrete Algorithms*. Submitted.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, den 29. Januar 2010

Matthias Bernt